University of Zurich[UZH]

Department of Informatics

Interactive Visualization of Large-Scale Geographic Data

Dissertation submitted to the Faculty of Business, Economics and Informatics of the University of Zurich

to obtain the degree of
Doktor / Doktorin der Wissenschaften, Dr. sc.
(corresponds to Doctor of Science, PhD)

presented by
Matthias Thöny
from Planken, Liechtenstein

approved in April 2017

at the request of
Prof. Dr. Renato Pajarola
Prof. Dr. Lorenz Hurni
Prof. Dr. Michael Böhlen

University of Zurich[UZH]

The Faculty of Business, Economics and Informatics of the University of Zurich hereby authorizes the printing of this dissertation, without indicating an opinion of the views expressed in the work.

Zurich, April 5, 2017

Chairwoman of the Doctoral Board: Prof. Dr. Elaine M. Huang

ABSTRACT

Geographic information systems progress strongly in a lot of different application domains. In addition, the gathering of geographic information is inevitable and large-scale geographic databases and server infrastructures need to handle this amount of data. Geographic data is growing in precision and complexity, so that it is necessary to explore and analyse datasets in an interactive way. Interactive visualization is key to explore geographic data sets and therefore indispensable for everyone using geographic data. In the following, new ways for interactive rendering techniques are presented to handle the challenges in nowadays large-scale geographic information systems.

The thesis starts with an introduction to geographic information systems followed by an analysis of requirements and challenges for geographic visualization systems and virtual globe systems. Furthermore, an introduction to the rendering pipeline is presented. To face the challenges of today's geographic visualization systems we introduce the *GlobeEngine* framework which enables a modular structure for rapid prototyping of geographic visualization applications and also contains all running prototypes of this work.

The main algorithmic contribution of this thesis are new rendering techniques, namely a new version of the *RASTeR* terrain engine, a innovative technique for rendering vector maps for interactive terrain and map visualizations and a novel graph bundling technique using vector maps as basis for bundling paths in a interactive 3*D* perspective environment.

Terrain visualization is the basis for an interactive 3*D* geographic visualization

system. However, it is hard for decision makers to clearly identify the best choice of terrain rendering algorithms. Therefore, this work provides an overview over terrain rendering requirements and existing terrain rendering solutions as well as their applicability to modern graphics systems. Furthermore, this thesis shows how *RASTeR* can be adapted to modern graphics hardware and a set of terrain visualization features, such as edge highlighting, ambient occlusion or terrain slope, aspect and flow visualizations to extend the capabilities of the existing terrain visualization.

Often, vector map visualizations are used on top of $3D$ terrain rendering. Interactive rendering of large-scale vector maps is a key challenge for high-quality geographic visualization software systems. This thesis contains a novel approach for the visualization of large-scale vector maps over detailed height-field terrains. This method uses a deferred line shading approach to render large-scale vector maps directly in a screen-space shading stage over a terrain visualization. The fact that there is no traditional geometric polygonal rendering involved allows our algorithm to outperform conventional vector map rendering algorithms for geographic information systems. A flexible clustered deferred line rendering approach allows a user to interactively customize and apply advanced vector styling methods, as well as the integration into a vector map level-of-detail system.

Dense line graphs and polyline maps are challenging for interactive visualization in geographic information systems. Bundling techniques are a common approach to reduce clutter and have successfully been demonstrated for the display of complex planar graphs. Previous techniques typically applied some forms of attraction or repulsion forces to bundle edges. In geographic visualizations, it is often necessary to take the semantic information into account and constrain path bundles to follow some reference network vector map. This thesis applies a novel method which uses geographic vector map reference information to route, visualize and simplify path bundles along their network paths in a constrained environment using adaptive B-splines.

The thesis is concluded by a summary and a future work section presenting future research topics.

KURZFASSUNG

Geografische Informationssysteme werden in verschiedensten Anwendungsgebieten verstärkt eingesetzt. Zusätzlich zu einer verstärkten Nutzung werden immer mehr geografische Daten erfasst und verarbeitet. Durch diese intensivere Nutzung von geografischen Informationssystemen ist es notwendig Infrastruktur wie Serversysteme und Datenbanken an die Herausforderungen anzupassen. Die Masse an geografischen Daten wächst ebenfalls durch mehr Präzision und Komplexität in der Erfassung und Verarbeitung. Um eine sehr grosse Masse an geografischen Daten zu analysieren, sind interaktive Werkzeuge und Visualisierungen notwendig. Somit ist das Gebiet der interaktiven Visualisierung ein wichtiger Forschungszweig für das Erkunden und Verstehen von grossen Daten und auch unerlässlich für die Arbeit mit geografischen Daten. Im Folgenden werden neue interaktive Bildverarbeitungsmethoden für die Visualisierung von geografischen Daten präsentiert. Mit Hilfe dieser Methoden lassen sich aktuelle Problemstellungen in der Bildverarbeitung von interaktiven geografischen Anwendungen lösen.

Die Einleitung in diese Dissertation erfolgt anhand einer Anforderungsanaylse von geografischen Informationssystemen. Es werden verschiedene Anforderungen und Problemstellung im Zusammenhang mit der Entwicklung von interaktiven Anwendung zur Visualisierung von geografischen Daten, wie bspw. eines interaktiven Globus, erörtert. Um interaktive Visualisierung zu verstehen wird zuerst auf den Bildverarbeitungsprozess in interaktiven Anwendungen eingegangen und anhand des Anwendungsframeworks, dass für diese Dissertation erstellt wurde grundsätzliche Aspekte erklärt. Die *GlobeEngine* ermöglicht interaktive

geografische Anwendungen zu erstellen und löst Probleme im Zusammenhang mit der Echtzeitdarstellung von Szenen in geografischen Anwendungen. Weiters ermöglicht das *GlobeEngine* Framework die schnellere Entwicklung von Prototypen für Visualisierungsprogrammen. Gleichzeitig werden alle entwickelten Prototypen im Zuge dieser Arbeit in diesem Programmpaket zur Verfügung gestellt.

Der Hauptteil dieser Arbeit erläutert neue Techniken zur Erstellung einer interaktiven 3*D* Visualisierung für geografische Daten. Als Erstes wird auf eine komplette Neuimplementierung des bekannten *RASTeR* Algorithmus eingegangen. Danach wird eine innovative Technik zur Darstellung von Vektordaten in einer interaktive Geländevisualisierung vorgestellt. Und zuletzt folgt eine neue Art der Bündelung von Graphen in einer interaktiven 3*D* Anwendung.

Geländevisualisierungen sind die Basis für interaktive 3*D* Anwendungen im geografischen Bereich. Trotzdem ist es immer noch sehr schwierig für Entscheidungsträgern die richtigen Verfahren zu identifizieren und umzusetzen. Dies hat zum Einen mit der sehr grossen Anzahl an Anforderungen zu tun und zum Anderen mit der Tatsache, dass ein Geländevisualisierungsystem sehr umfangreich in der Entwicklung ist und ungern grössere Überarbeitungen gemacht werden seitens der Hersteller. In dieser Arbeit wird ein Einblick in die Anforderungen für interaktive Geländevisualisierungsalgorithmen gegeben und existierende Lösungen genauer anaylisiert. Weiters wird anhand des *RASTeR* Algorithmus gezeigt wie ein solcher Algorithmus auf moderne Grafikhardware angepasst werden kann. Zudem werden neue Methoden für die Visualisierung von Gelände gezeigt wie bspw. Kantenhighlights, Umgebungsverdeckung und die Visualisierung von Gefälle oder Hangrichtung.

Oft werden in bereits existierende Geländevisualisierungen geografische Vektordaten eingebaut. Interaktive Visualisierung von grossen Vektordaten ist eine wichtige Problemstellung für ein qualitativ hochwertiges Visualisierungsystem. Diese Dissertation enthält einen neuen Ansatz für die Visualisierung von Vektordaten auf einer bestehenden Geländevisualisierung. Die Methode basiert auf der Idee, dass die Linienschattierung in einem zweistufigen Verfahren erfolgt. Zuerst wird das Gelände erfasst und verarbeitet und im zweiten Schritt werden die vorhandenen Bilddaten genutzt um die korrekte Position der Vektordaten zu errechnen. Im Gegensatz zu vorhergehenden Methodiken wird keine traditionelle Geometrieverarbeitung für die Vektordaten ausgeführt. Dadurch ist es möglich mehr Bilder pro Sekunde zu erzeugen als bisher verwendete Algorithmen zur Verarbeitung von Vektordaten. Dieser flexible Ansatz ermöglicht es ebenfalls, dass Vektordaten interaktiv benutzerspezifisch gestaltet und angepasst werden können. Ebenso ermöglicht der neue Ansatz eine Integration in bestehende Systeme mit anpassbaren Detaillierungsgrad in der Visualisierung.

Dichte Graphen und Graphnetzwerke sind ebenfalls bekannte Herausforderungen für interaktive Visualisierungsanwendungen. In geografischen Visualisierun-

gen werden dichte Graphen oft für Verkehrsinformationen oder Migrationsströme verwendet. Bündelungstechniken sind gängige Verfahren um die Information von dichten Graphnetzen hervorzuheben. Es wurde bereits mehrmals erfolgreich gezeigt, dass der Informationsgehalt bei der Verwendung von Bündelungtechniken mit komplexen planaren Graphen steigt. Vorhergehende Methoden verwenden typischerweise eine Kombination von Anziehungs- oder Abstossungskräften für die Bündelung des Graphnetzwerks. In geografischen Informationssystemen ist es oft notwendig, Informationen über die Umgebung des Graphnetzwerk zu berücksichtigen. Darum ist sinnvoll die Bündelung von Graphen anhand eines weiteren Netzwerks, bspw. eines Strassennetzes zu erstellen. In dieser Dissertation wird ein Algorithmus vorgestellt der vorhandene geometrische Informationen aus Vektordaten zur Bündelung und Visualisierung von Graphdaten verwendet. Für die Darstellung von Strömungen aus einem Graphnetzwerk in einer $3D$ Visualisierung werden sogenannte B-Splines verwendet.

Die Dissertation wird durch eine Zusammenfassung der Ergebnisse und einer Übersicht über zukünftige Forschungsthemen abgeschlossen.

ACKNOWLEDGMENTS

During my PhD, I had a lot of support from various people. Special thanks goes to my advisor Renato Pajarola, who is always patient and supportive, but also critical when it is necessary. I am very thankful for the opportunity to profit from Renato's unique knowledge and to conduct a PhD at the University of Zurich.

Furthermore, I want to thank all former and current members of the VMML team for listening to provoking theories, weird new ideas, radical opinions about software engineering and challenging climbing sessions. Special thanks goes to the favorite postdocs, namely Oliver, Markus and Enrique for their ongoing support and help. I also want to mention the team of the *Atlas of Switzerland* and there especially, Remo Eichenberger and René Sieber. They supported me with a second workplace, many challenging objectives, helpful discussions and a unique view on software development in graphics.

For proofreading and correcting parts of this thesis, I want to thank Enrique Paredes, Markus Billeter, David Steiner, Valeria Botka, Edi Risch, Gregory Wyss and Armin Moosbauer. In addition, I am very grateful for code contributions to the *GlobeEngine* from Enrique Paredes, Markus Billeter and students projects, namely Alireza Amiraghdam, Florentin Liebmann, Marco Bonzanigo, Henry Raymond, Simon Ruesch, Matthias Nötzli and Pascal Forny.

At last, I also want to thank the Federal Office of Topography Swisstopo as well as the Landesvermessungsamt Feldkirch, Austria, for providing multiple data sets and helpful cooperations for this project.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Figure 1.1: *Tabula Peutingeriana shows a schematic map of the road network of the roman empire.*

Humans like visualizations. The need to abstract information in visual form and sketch things is as old as humanity itself. Visualizing information in form of maps was always an important part in human history. Maps are done because humans have a communication problem when the amount of information to transfer is too complex. We could express everything in words, but the result will be error-prone. There is a desire to make communication fail-safe and easier by introducing abstract information and visualizations for the transfer of knowledge. Maps are examples for such visualizations, because maps order information spatially and often also temporally. In addition, dividing significant information from

nondescript information makes it possible to focus on a specific topic of information without further visual distraction. Maps are often used with this principle. A nice example for this is the *Tabula Peutingeriana* showing the complete street network of the roman empire.

This map had the purpose to guide a traveler trough the complex network of roman streets. It shows major cities and forts but it also contains information about horse stables so that the observer is able to improve route planning and traveling times. The abstraction of the town shapes and the simplified drawing of streets is intentionally understandable to everyone at first sight. On the basis of this example we see that the visualization of geographic information is a very basic form of communication and collaboration between single persons and groups of persons. The demand on this basic form of communication remained the same over centuries and is still the driving force behind all digital and non-digital maps.

1.1 Learning based on Visualizations

When two persons meet, usually they define a time and a certain location as a meeting point. Often, people choose known places to meet. However, finding the right way to these meeting points is often tricky and nowadays it is done with help of digital map tools. Usually, it is possible to set an initial position and destination location. After a quick search, the software shows the user a map with several opportunities how to reach the meeting point. Most people are not aware that a common mathematical problem is associated with this task. The challenge of finding the shortest path in a graph network. What the software system does, is to solve a shortest path algorithm based on the person's input and returns the shortest path which seems to be the best option for this case. However, the shortest path is not always easy to determine and sometimes it needs more information such as the choice of a transportation vehicle. The system recognizes this input and offers the user different information about every transportation device. Furthermore, the system might offer additional or alternative paths based on the user's input or it offers the possibility to input additional way points and then further refines the results.

What intuitively happens is that the user iteratively tackles the challenges and learns based on a visualization. In this simple case, the challenge is described as the question how to reach the destination and the map is the appropriate visualization for this. As soon as the user understands or evaluates the visualization of the first shortest path result, knowledge is created and this immediately leads to a new question, namely whether the proposed shortest path to the destination is the optimal path for the user.

The described iterative learning process is a simplified example for what sci-

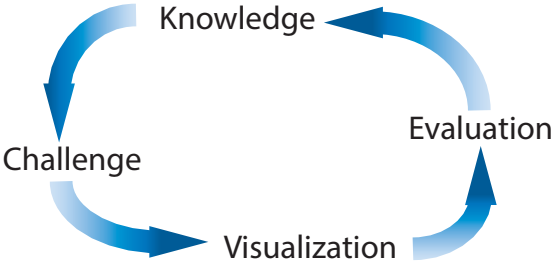


Figure 1.2: *The process of iterative knowledge acquisition. Challenges and problems often require support of visulizations. Evaluating these visualizations gives insights and leads to new knowledge. Thus new problems can be formulated.*

entists call visual analytics in geographic information systems (GIS). What happens is illustrated in Fig. 1.2. This iterative learning process is the motivation for providing better visualizations. With a better understanding of the challenge, we are able to come up with more precise and better solutions. With good visualizations, we gain deeper insight into a problem domain so that on the one hand we can solve problems where the solution was unclear before and on the other hand we can elaborate more challenges and questions which were not recognized beforehand.

The field of visualization as a subfield of computer graphics has the target to improve the process of understanding data by improving the visual representation of this data to enhance the communication quality. An important factor of this visual representation is the amount of interaction a system can provide. When a system reacts immediately on user input the above learning cycle is accelerated and the problem exploration is improved. This also applies to geographic information systems, where the basic interaction possibilities on a map such as zooming or navigating improve the user perception of the data space and help to understand in which context the data exists. The collection of these interaction processes and operations on different data can be summarized in a visualization pipeline.

1.2 Visualization Pipelines

The core of a visualization system is the modeling of the visualization pipeline. In a visualization pipeline, the raw data material is transformed in different steps to achieve a final displayable image. A common model for a multivariate multi-

dimensional visualization pipeline is shown in Fig. 1.3. Typically, raw data sets are gained from an input source or from a database and therefore such data sets contain incomplete or incorrect records. The first step is performing a data analysis process to decide what part of the data should be considered to influence the visualization, such as empty value filling or merging small data sets together. Typically, this step is done only once per data set using scripts and toolboxes which require some technical or domain experience. In the above mentioned shortest path example, this step would be equivalent to the preparation of the map data so that shortest path requests can be executed on a mapping server.

After the data analysis step, the visualization data is ready to be used in a visualization tool, often a customized viewer software. With this tool, a user centric filtering process is applied. In our introductory example, this could be zooming or selecting input data, such as the search for a certain destination or highlighting a certain path object. With help of the user's input, the visualization system knows which objects are important. The system can apply this knowledge to highlight interesting features of the data in a visually interesting form. This is done in the mapping step. Another example for this is the assignment of a certain color to the highlighted path. Furthermore, the geometric representation (points, lines, polygons) is chosen in this step as well. The outcome of this processing step can be described as the conceptual visualization object prepared for the final rendering process. The rendering process creates the final displayable image of the data interpreting the input from the former processes. This concludes the visualization pipeline as shown in Fig. 1.3. Designing visualizations and designing visualization pipelines is currently an active field of research. The book [Munzner, 2014] gives a deeper insight into visualizations and visualization pipelines.

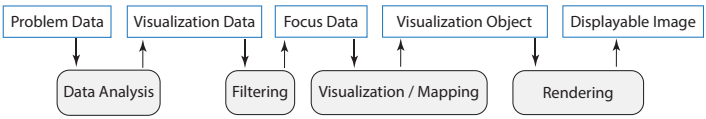


Figure 1.3: *The extended dataflow model for a multivariate multidimensional visualization pipeline as described in [dos Santos and Brodlie, 2004]. It shows the states of data (top row) and the processes on the data (bottom row) within a visualization pipeline.*

The core contribution of this thesis is an implementation of such a visualization pipeline for large-scale geographic data. Before delving further into this topic in Chapter 2, we will discuss specific challenges and problems of visualization pipelines for geographic information systems.

1.3 Virtual Globe Visualizations

Visualization systems for geographic or spatial information have specific problems and challenges which are highly depending on the application requirements. For the purpose of this thesis we look at specific visualization systems called virtual globes. Commonly known representatives in terms of applications are *Google Earth*[1], *NASA Worldwind*[2] and in terms of frameworks are the *osgEarth*[3] and the *Cesium* framework[4]. The survey paper about digital earth systems shows the current state-of-the-art in this domain [Mahdavi-Amiri et al., 2015]. The book [Cozzi and Ring, 2011] describes a typical construction of a virtual globe system. Other works related to virtual globe challenges and goals can be found in [Hildebrandt and Döllner, 2010] or [Christen, 2008]. Virtual globes can have many different applications for visualizations. In the past, virtual globe software packages were used to visualize:

- General maps: e.g. street maps, borders, other geographic feature data
- Imagery: e.g. satellite images, infrared images or other image information
- Political information: e.g. voting results, population density
- Climatological information: e.g. temperature information, wind or underwater streams
- Geological information: e.g. rock formations, molasse information
- Hydrological information: e.g. water quality, ground water information
- Social network information: e.g. interesting trails, photo spots, animated videos paths

Depending on the specific application there can be many different challenges concerning a virtual globe framework. From a computer graphics point of view, the main challenges for a virtual globe software can be categorized into four categories, namely: Data Handling, Geometric Representation, Rendering and Visualization aspects. Fig. 1.4 shows an illustration of challenges which can influence the development of a virtual globe system.

Data handling challenges are problems which are associated with the data source. Raw data can often be a problem for a visualization software, because it is unstructured in terms of formats, sources or ordering. Therefore, typical data handling problems are filtering and preprocessing. In a lot of visualization and virtual globe systems, the usage of large-scale data sets is restricted because

[1] https://earth.google.ch/
[2] https://worldwind.arc.nasa.gov/
[3] osgearth.org/
[4] https://cesiumjs.org/

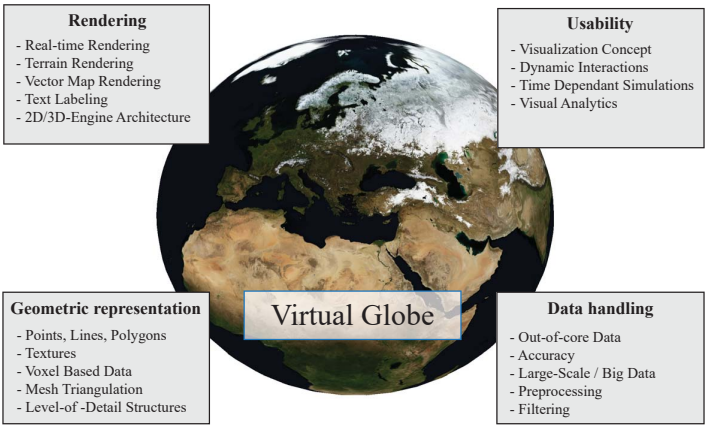


Figure 1.4: *An overview on challenges when developing a virtual globe system.*

systems handling large-scale data need more advanced algorithms than usual visualization system. Often, such systems need specific out-of-core techniques to handle large-scale data sets at a certain accuracy.

The geometric capabilities within a visualization system can have significant impact on a virtual globe. Graphics systems often require a dedicated and optimized rendering structure to interact with the *GPU* (Graphics Processing Unit). Commonly known virtual globes or other visualization pipelines work with triangle meshes, textures, points, lines, polygons or voxel based data sets. The challenge is usually the choice of the right geometric representation for a certain combination of data type, visualization type and the subsequent algorithmic complexity to transform the data into this geometric representations. Often, an important challenge is the usage of level of detail data structures to handle large data sets for texture-, voxel- or mesh triangulation structures. A detailed description on metric data structures can be found in [Samet, 2005].

Rendering as the creation of a single image is an important task for a visualization system. Common challenges in rendering a virtual globe system are the rendering of terrain, vector map information and related text information. For interactive visualization the main challenge is to achieve 16 milliseconds per rendered image or 60 frames per second. The human eye can distinguish between $25 - 30$ pictures per second. Depending on digital displays and the type of dis-

play more images per second are necessary. Therefore, 60 frames per second is an agreed objective in the computer graphics community for every interactive rendering system. Rendering engines can be designed in different ways but usually there is a difference between the development of an interactive system supporting only $2D$ data sets and a system handling a fully interactive perspective $3D$. In the following we always assume the latter case.

The usability aspects in a virtual globe software contains several challenges. This can be on one side dynamic interaction with the globe object like navigating with a perspective $3D$ camera or selecting parts of a data set but also visual analytic tools which have to be designed in an understandable but still usable way. Domain specific users often require another access to the interface than other users. Moreover, the usability aspect is dependent on the data sets and the capabilities of the systems. For example, time-dependent $3D$ visualization are usually more complex when it comes to interaction capabilities. In the following section, some of these challenges will be described more in detail to understand the specific challenges of this thesis project and its contributions.

1.4 Challenges

As shown in Fig. 1.4, there are many challenges during the development of a virtual globe system. This thesis project focuses on the following three main challenges:

- **Terrain Visualization System**: The main goal is to create an interactive terrain visualization system which is comparable to existing virtual globe software packages. The main challenge is the handling of large amounts of height information. To give an example, the SwissALTI3D data set contains the height information of Switzerland on a resolution of $1m$ and has a data size of $60GB$. The imagery of Switzerland in a resolution of $50cm$ has around $700GB$. To explore this information in an interactive $3D$ viewer, it is necessary to implement a dynamic out-of-core Level-of-Detail system. Several techniques already exist to solve problems such as dynamic height field triangulation, height field and image compression or client-server architectures. But there are several untended aspects in terrain rendering. Examples are aspects of effective data visualization, such as visualization of terrain properties like slope, aspect or flow direction, the applicability of advanced rendering algorithms like ambient occlusion, large-scale shadow mapping, alpha compositing of multiple image or heightfields or the seamless transition of different heightfield repositories in one viewer. All these aspects make the construction of a terrain visualization system to a complex problem. Chapter 3 describes this more detailed and provides a way

to combine these challenges to an out-of-core terrain visualization system which is flexible enough to manage large-scale data sets and is able to apply different kinds of visualizations.

- **Interactive Large-Scale Vector Map Rendering**: Geographical features are expressed as vector maps and can contain different types of geometric data, usually points, lines or polygons. The main goal for this work is to provide large-scale vector map rendering within the interactive terrain visualization. Usually, vector maps contain a database of information associated to their geometric types. Therefore, geometric points, lines or polygons can be connected with a set of multiple attributes, similar to an entry in a database row. This implies that a vector map can have the same geometry, but it needs a varying visualization technique. Thus, vector maps might need more interaction possibilities for the user in terms of exploration capabilities and customization. A street map for the purpose of driving in a navigation system needs another visualization than the same street map for tourist purposes even if it uses the same data set. Another aspect of rendering vector maps in perspective $3D$ environments is the correct placing and mapping of the vector data on top of the terrain. Several rendering errors can appear with current techniques. The method presented in Chapter 4 and published in [Thöny et al., 2016] is able to render large amounts of vector map data on top of the terrain visualization system presented in Chapter 3 and compares the results against existing state-of-the-art techniques.

- **Graph Bundling of Large Graph Data Sets for GIS**: In general, vector maps can be described as large graph networks, for example a street network. These graph data sets describe a geometric mapping, such as existing infrastructure. But vector maps are not restricted to this property. A spatial graph network can also describe political or economical information and connect spatial information to it. The spatial connection to such kind of information can have significant impact to the user's learning process. However, large graph networks suffer from cluttering artifacts when visualized directly. Graph bundling describes a set of filtering techniques for dense graph networks to improve the visual information given by this graph network. The main goal of this work is to find a way to apply graph bundling in perspective $3D$ environments so that bundles are routed according to $3D$ feature data. Challenges for graph bundling algorithms are: improving the visual quality by reducing cluttering, improving the information quality by making the process of bundling interactive and adjustable for the user, and integrating such graph visualizations into existing perspective $3D$ environments. Further details about this graph bundling challenges can be found

in [Thöny and Pajarola, 2013]. Chapter 5 shows how to achieve the above mentioned goal for bundling large-scale graphs in a 3D perspective environment using geographically referenced information.

The above collection of challenges provides an overview for the following chapters. As illustrated in Fig. 1.4, several other problems might appear when implementing an interactive rendering system for large-scale data sets, such as handling large amounts of data in an out-of-core system, numerical precision issues or the preservation of interaction possibilities through the visualization pipeline.

1.5 Contributions

This thesis project contributes significantly to the field of geographical scientific visualizations. GIS software can profit from the presented techniques in terms of rendering speed, interaction quality and visual rendering quality. The contributions of this work can be summarized as follows:

- **Terrain Rendering:** The terrain rendering system in this thesis is based on the former works related to terrain visualization and triangulation algorithms [Goswami et al., 2010], [Bösch et al., 2009], [Pajarola and Gobbetti, 2007] and [Gerstner, 2003]. The main contribution is a complete new interpretation of the *RASTeR* algorithm combined with a flexible out-of-core visualization system. The terrain rendering system allows different texture levels and different height field levels as well as connections to different out-of-core terrain repositories over a tile map service structure. Additionally, examples for real-time *GPU* terrain feature visualization are shown such as the visualization of slope and aspect as well as visual effects such as ambient occlusion. A publication about future trends in terrain visualization can be found in [Thöny et al., 2015].

- **Deferred Vector maps:** An integral part of this thesis is the rendering of large-scale vector maps with and without terrain. Deferred vector maps solve the problem of combining vector map data with terrain rendering data in a visualization with a minimum of rendering artifacts within a perspective 3*D* environment. Additionally, a method is presented to solve accrued rendering artifacts using an alpha blending shader implementation. The results of the technique are evaluated by comparison with standard rendering techniques. The results of this work have been published in [Thöny et al., 2016].

- **GPU Chart rendering:** Chart rendering of political or economical information is a traditional application of vector maps in geographic information science. Our point based GPU implementation of standard chart visualizations opens the possibility to display a large amount of dynamically adjustable charts within a visualization pipeline. A short article about the results can be found in [Thöny and Pajarola, 2014].

- **Constrained Path bundling:** Graph bundling is a well known visualization technique to visually improve cluttered graph networks. In this thesis, a method to generate vector map constrained path bundles is presented. These bundles are adjusted along a traffic or meta data network to improve the visual information of a cluttered graph in a perspective $3D$ environment. To achieve a terrain independent visualization, path bundles are routed around mountains under the constraint of underlying vector map information. The evaluation is done by comparing it with earlier graph bundling approaches. The outcome of this work is published in [Thöny and Pajarola, 2015].

- **The GlobeEngine Framework:** The thesis contribution and publications are unified in a visualization framework called the *GlobeEngine* framework. The framework itself was also used in several student projects about interactive rendering related to geographical or astronomical visualizations. The main goal of the *GlobeEngine* framework is to provide a prototyping environment for visualization projects working with large-scale data sets.

1.6 Dissertation Overview

This dissertation is structured in 6 chapters. Chapter 1 gave a short introduction to geographical visualization. Furthermore challenges and contributions of this thesis were outlined. Chapter 2 gives an overview over the goals and capabilities of the *GlobeEngine* system as well as a description of the concrete visualization pipeline and the system architecture. In addition, it makes the connection to the following more detailed techniques incorporated in the *GlobeEngine* framework. Chaper 3 introduces the reader to the terrain rendering part of the *GlobeEngine* and gives a detailed explanation how the terrain rendering system works. Alternative terrain rendering methods are discussed as well. Chapter 4 presents vector map visualization techniques. Furthermore, it provides a comparison between our deferred vector map visualization technique and traditional vector map rendering methods. Chapter 5 presents vector map constrained path bundles along with a detailed algorithmic implementation. In addition, an evaluation against other methods and further improvements are presented. Chapter 6 concludes this thesis with a summary statement and gives directions and plans for future work.

CHAPTER 2

THE GLOBEENGINE VISUALIZATION SYSTEM

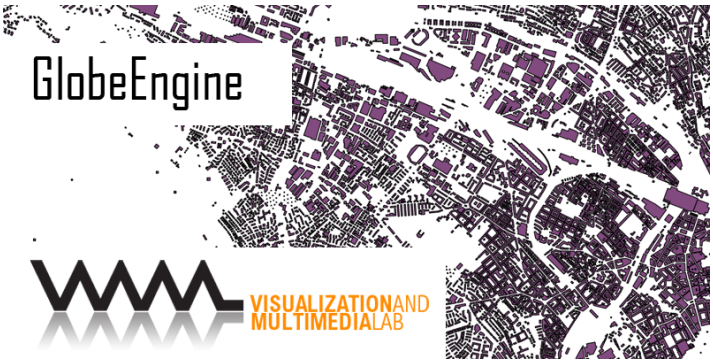


Figure 2.1: *The GlobeEngine Logo.*

2.1 The GlobeEngine Framework

The development of the *GlobeEngine* framework started in 2011 with this PhD project. The reason to start the development of this framework was the need for unification of former research in the field of terrain rendering done by the VMML group. Furthermore, future projects and developments of this PhD project for rendering large-scale vector map visualizations should be integrated as well. The main goals of the system can be summarized as follows:

- Providing a redesign of the existing implementation of *RASTeR* and develop a new terrain engine using the *RASTeR* terrain rendering algorithm.
- The design and development of a large-scale vector map engine to visualize vector map data sets with different rendering techniques.
- The implementation of visual analysis techniques, especially graph bundling techniques within the *GlobeEngine* framework.
- The design of a framework to provide the *VMML* research group and the students associated with VMML a unified visualization framework for rapid prototyping of large-scale GIS visualizations.
- The implementation of rapid prototypes for papers or external projects such as the *3D Atlas of Switzerland* to test rendering techniques or algorithms within a controlled and simplified *OpenGL* environment and without relying on third party render engines.

2.1.1 System Architecture

The *GlobeEngine* framework architecture is shown in Figure 2.2. The framework is designed for cross platform development on Windows, Mac OSX, Linux Debian and Linux Ubuntu. These platforms are the core development platforms for all VMML projects as well as widely used in the scientific visualization community. Integration and interaction between other VMML projects such as *Equalizer* [Eilemann et al., 2009] are important future plans and therefore architecture decisions were made to make this possible in the future. Despite the recent trend for high level languages like JavaScript or Ruby, the core focus of this system is interactive rendering and especially hardware *GPU* programming. To get the maximum performance out of a visualization system in combination with cross platform functionality, C++ is still the required language. There are a lot of existing tools and third party dependencies available to work together with C++. The most important libraries used in this project are *QT* for user interfaces and many low level system interactions such as threading, networking or window handling, as well as *OpenGL* & *OpenCL* for graphics driver interaction and multi purpose *GPU* computing.

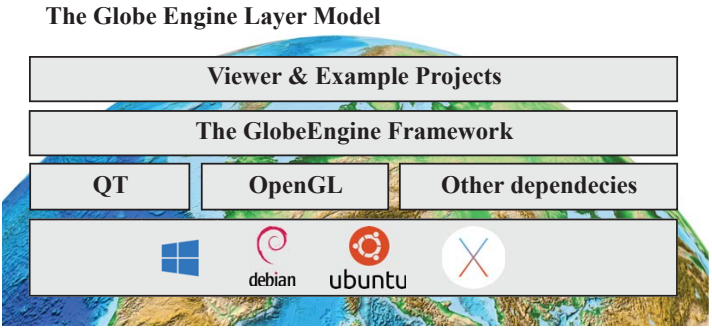


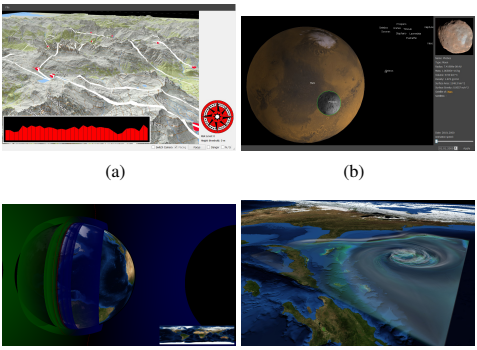
Figure 2.2: *The GlobeEngine Layer Model.*

The framework builds up on *CMake* to organize cross platform dependencies and development environments such as *VisualStudio*, *XCode*, *QTCreator* and other platform specific build tools. The framework itself contains several packages, along with concrete viewer projects which will be discussed in the following sections. Fig. 2.4 gives an overview over the *GlobeEngine* package structure.

2.1.2 Package Overview

The package overview shows the actual state of the *GlobeEngine* framework. During the development of the *GlobeEngine* framework several viewer applications were programmed by our undergraduate students using this framework. The long term effect of integrating these student projects into the *GlobeEngine* framework is that these prototypes share a common source base so that student projects do not get lost immediately after the submission of students, because their know-how leaves the group.

This maintenance of student or PhD projects is often ignored, because it seems like a waste of resources. As a consequence, existing source code is often not reused in newer projects by PhDs or master students because the know-how is lost and testing existing code fails because it was not maintained properly. Especially, updating viewer projects for new operating systems or new graphics hardware can take many work hours if the software was not maintained over the years. In *GlobeEngine*, the maintenance of student projects is done immediately during development, e.g. porting to a new operating system. All viewer projects share a common source base so that the code changes are much smaller and therefore



(a) (b)

(c) (d)

Figure 2.3: *Example screenshots from GlobeEngine student projects. (a) Hiking Viewer, (b) Exoplanet Viewer, (c) Atmosphere Viewer, (d) Planetary Viewer.*

easier to adapt. In addition, the teaching aspect for students is important. We want the students to get hands on experience with low level graphics code. At the same time, we want them to adapt existing visualization techniques and clean code styles by introducing them to real-world visualization packages.

Our experience in this project shows that student projects and student code can be integrated well into an ongoing research project if the student is able to use the same code base as the ongoing research project. Fig. 2.3 shows some examples of such student viewers. For the following Chapters only the relevant subset of packages will be described:

- **Core Modules**: This modules contain the core packages of the *GlobeEngine*. The indiviual packages contain rendering structures, spatial data structures, graph data structures, window handling as well as an simple example viewer for testing purposes and as template for student projects.
- **Terrain Modules**: The terrain rendering module contains the code for the *RASTeR* algorithm as describes in Chapter 3 and the code for spatial messaging for interaction with the tile-based storage systems. It also contains a simple example viewer only capable of rendering terrain.
- **GIS Modules**: The GIS modules contains all source code specifically related to vector maps visualizations (in the coding context this is described as features) within the *GlobeEngine*. It also contains the *GlobeEngineViewer* project, which is the main application build for this thesis project.

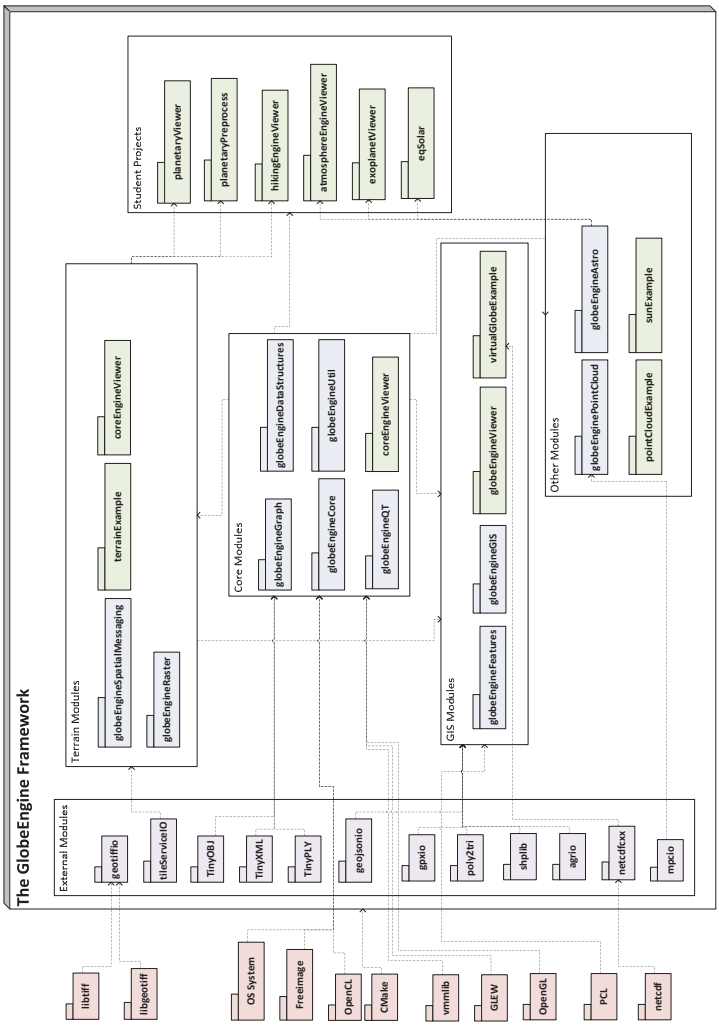


Figure 2.4: *The GlobeEngine Framework. Packages in red are external dependencies. Violet packages are internal or external modules integrated into the GlobeEngine build structure. Green packages are GlobeEngine viewer projects. Blue packages are GlobeEngine modules.*

2.1.3 The GlobeEngine Visualization Pipeline

The *GlobeEngine* visualization pipeline can be seen as a collection of individual visualizations designed for different purposes but resident in the same software application prototype. Every individual visualization is designed according to the design principles described in the book [Munzner, 2014] and every interactive visualization is based on three questions shown in Fig. 2.5:

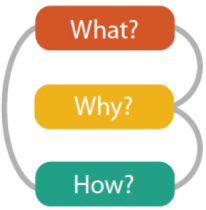


Figure 2.5: *Illustration of an iterative process for creating visualization. Image courtesy of Tamara Munzner from [Munzner, 2014]*

An overview over the interactive visualization pipelines in the *GlobeEngine* framework is given by Fig. 2.6.

The purple pipelines show visualizations of elevation and digital imagery which is described in Chapter 3. The main purpose of these visualization types is to provide an insight into the 3*D* nature of large-scale raster data sets like height maps or satellite images to visualize height differences in a natural way, which is hard to see on 2*D* images. The red lines show visualization pipelines for different kinds of vector maps. The main purpose for these visualizations is to display the content of the vector maps inside a perspective 3*D* terrain rendering environment in an illustrative way. But there are some examples where the visualization pipeline changes with the purpose of the visualized item. For example, point data sets can be visualized as locations but, alternatively, the same point data set can be visualized as charts of a voting result on these locations. The goal of the *GlobeEngine* framework is to provide the user with different solutions for the question on how the data should be visualized. Different visualization pipelines are available in the *GlobeEngine* supporting old and new visualization and rendering techniques. Another example for these features are border data sets. Border data can either be used as lines to mark borders in the perspective 3*D* view or it can be used as polygons to visualize statistical data about the economical growth of a region. The next chapters will go into detail about the new visualization techniques used in the *GlobeEngine* framework.

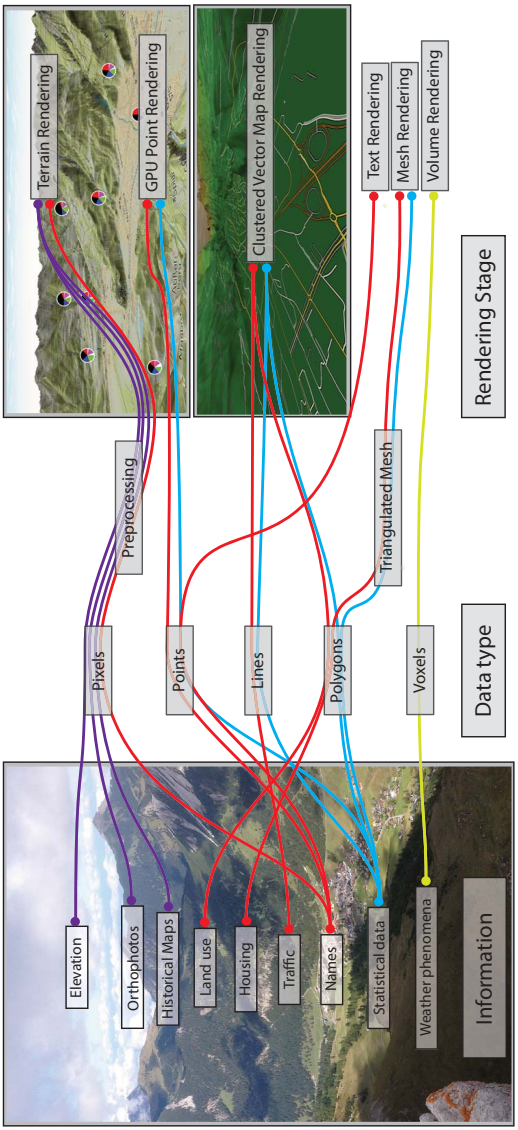


Figure 2.6: *The GlobeEngine Visualization Pipeline.*

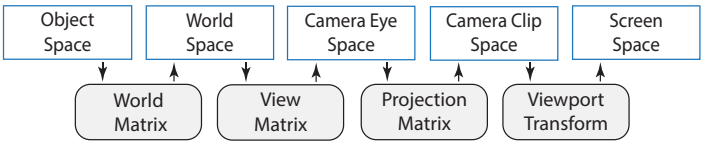


Figure 2.7: *Illustration of spaces in the traditional rendering pipeline.*

2.1.4 The GlobeEngine Rendering Pipeline

Traditional rendering pipelines are built by the pattern illustrated in Fig. 2.7. A geometric object, e.g. a triangle, exists in object space and is transformed in several steps, first into world space, then to camera space and finally to clip space. The last step is the transformation into screen space. Usually, these steps are expressed using matrix multiplications. More details about these transformations can be found in several sources, e.g. in [Akenine-Moller et al., 2002].

Graphics APIs such as OpenGL[1], DirectX[2] or Vulkan[3] provide programmers an API to access the graphics hardware and manipulate specifically the implementation of the rendering pipeline. Typically, the programmer can influence parts of the pipeline either over API commands or by shader programs. In recent years the complexity and the amount of APIs was growing. The *GlobeEngine* framework uses the capabilities of the OpenGL API from version 4.5 as illustrated in Fig. 2.8.

A traditional visualization program implemented in the *GlobeEngine* framework setup defines first vertices and primitives such as lines and triangles and uses vertex buffers to store the geometric information on the *GPU*. These vertices are then transformed into camera clip coordinates inside the vertex shader program. In the following rasterization step, primitives are converted into sets of fragments. A fragment can be defined as a screen pixel with depth information. Redundant fragments will be discarded and for every fragment a shader program is executed. The output of this fragment shader is then stored in a framebuffer. In a simple pipeline, this framebuffer will be the output of the graphics hardware to the screen but in more advanced graphic engines, it can be used as input for further steps.

State-of-the-art rendering engines use several instances of framebuffer objects to create a final image on the screen. The purpose of framebuffer objects can vary depending on the application purpose. A common example is the overlay of an user interface (UI) over a 3*D* scene. This technique is often used in game engines. An example can be found in Fig. 3.17. The idea is to render UI elements in a

[1] https://www.opengl.org/
[2] http://msdn.microsoft.com/directx/
[3] https://www.khronos.org/vulkan/

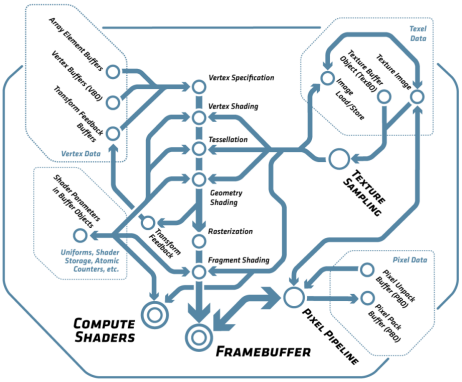


Figure 2.8: *Illustration of the OpenGL pipeline version* 4.5. *Image courtesy of NVIDIA.*

separate framebuffer and the 3*D* scene in a second framebuffer. Then, combine both in a separate composition step. This separation has several advantages. First, the UI is painted only after changes in the UI. This can be a significant difference because user interaction does not require 60 frames per second like the 3*D* scene. Second, the UI framebuffer can be used as a mask to avoid unnecessary tasks for pixels which are covered by the UI. Rendering steps for these pixels can be avoided. And third, when working with multithreaded APIs, the user interface can be generated by another rendering thread or another graphics card and be reused by the main rendering thread.

The *GlobeEngine* rendering pipeline uses multiple framebuffer objects to implement a deferred shading pipeline as described in Sec. 4.2. Usually, deferred shading pipelines are used to reduce the calculation effort for fragment shaders and apply post processing effects to the scene. A typical effect is ambient occlusion, which is described in Sec. 3.5. The main idea behind deferred shading is to store all information about a scene within a framebuffer object and therefore as a 2*D* image with depth information. When applying an effect, the calculations do not have to be done once per pixel for every primitive but rather once per pixel of the framebuffer. The downside is the limited availability of scene information, because calculations, such as lighting, have to be performed based on a 2*D* image with depth information. Additional details about the usage of deferred shading pipelines can be found in the articles [Shishkovtsov, 2005] and [Koonce, 2007].

CHAPTER 3

TERRAIN RENDERING

Figure 3.1: *Example rendering of the Rhine Valley.*

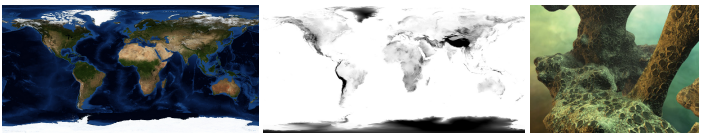
3.1 Introduction

Terrain rendering is an essential part of a virtual globe software package. The term terrain rendering encompasses all processes and stages in a visualization pipeline related to the visualization of terrain information. This includes not only the rendering of digital elevation information but also the visualization of additional geospatial information.

Digital terrain information can be divided into $2D$, $2.5D$ or $3D$ information. Usually, $2D$ information are pixel maps, flight images or satellite images such as the one in Fig. 3.2(a). Another type of raster based image information is the $2.5D$ elevation model in Fig. 3.2(b). Such $2.5D$ elevation models are also called digital elevation models(DEM), digital terrain models (DTM), elevation maps or height fields. This type of information is the common basis for nowadays terrain rendering algorithms.

In a DEM essentially every pixel of the image is interpreted as a corresponding height value. In principle, it describes a discrete $2D$ function $h_{i,j} = f(x_i, y_j)$. By interpreting the values as heights, the terrain receives a $2.5D$ character as we can see in the teaser image, Fig. 3.1. However, this $2.5D$ information is also the main limitation of todays terrain rendering systems, because it is not possible to express all occurring shapes of real terrain. To display all forms of $3D$ terrain, e.g. Fig. 3.2(c), it is necessary to acquire the terrain information in $3D$ as well. The main difference is that a complete $3D$ representation allows the visualization of caves, overhanging $3D$ structures like bridges or tunnels as well as subsurface structures. Recent research in $3D$ laser scanning open the possibility to acquire $3D$ structures in reasonable time. It will only be a question of time until these systems are used to systematically enrich the $2D$ information with $3D$ point clouds. However, the focus of this work is limited to $2.5D$ data sets.

$3D$ terrain rendering makes large-scale elevation information more accessible for the user by providing a more friendly and realistic impression. For daily tasks, a perspective $3D$ rendering might not be the optimal tool. For example, map search is better performed in $2D$, because abstracted $2D$ maps are easier to understand for humans than complex $3D$ environments. However, terrain render engines allow to map real $3D$ space as proper $3D$ information. Therefore, it is possible to simulate $3D$ worlds for education, entertainment, exploration, and planning purposes where a $2D$ map is not enough. With realistic terrain rendering, we are able to create realistic pilot training facilities and new ways for cultural story telling, such as creating new worlds in games. Terrain rendering is a key technology to communicate such experiences and stories. With upcoming virtual reality hardware, it will be possible to create immersive experiences for entertainment, training and research. In addition, terrain rendering is a key technology for planetary exploration, such as the Mars-Rover mission. It is already possible to



(a) Blue Marble Example (b) Gebco Example (c) 3D Example

Figure 3.2: *(a) Blue Marble Texture published by the NASA Blue Marble Project*[1]*. (b) Gebco digital terrain map. (c) Example for a* 3*D cave showing also overhangs and* 3*D structures.*

create complete 3*D* real-time simulations from satellites and rovers on extraterrestrial objects, and therefore, explore extraterrestrial planetary surfaces from a home computer.

3.2 Related Work

Terrain rendering is a well known area of research and for many aspects of a terrain rendering system there exist good solutions. In this section we discuss related work and the theoretical background for terrain rendering systems. In general, terrain rendering research can be divided into the following categories: triangulation algorithms, level of detail structures, client server architectures, compression algorithms and applications of terrain analytics. Because this chapter is focused on terrain visualization we will not go into detail about terrain modeling or procedural modeling algorithms. An introduction to the principles of terrain modeling can be found in the book [Li et al., 2005]. A hands on explanation for building a 3D terrain rendering engine is given in the book [Cozzi and Ring, 2011]. The most recent survey about state-of-the-art methods can be found in [Mahdavi-Amiri et al., 2015].

3.2.1 Terrain Rendering Pipelines

There are two different ways for top level designs of terrain rendering pipelines. The traditional approach uses a *DEM* to generate a triangulated mesh and will be discussed in detail in this chapter. The second approach uses a *DEM* inside a ray casting engine as proposed in [Dick et al., 2009a] and [Dick et al., 2010]. The main difference between them is that ray casting is a highly pixel based method and does not need any detailed mesh structure from a *DEM*. The papers show that this method is able to perform terrain ray casting in real time. As usual for

ray casting algorithms, the hardware requirements for the GPU in terms of performance are high. The reason is that ray casting demands a lot of processing power from the fragment shader, especially for large screens. Therefore, it might be a challenge to achieve high frame rates for HD or $4K$ screens, or on systems with limited hardware capacity such as embedded systems. It is highly likely that this limitation is omitted in the future, because of increasing performance on graphics hardware and new techniques for rendering. For example, it is possible to compute safety shapes from *DEM* information to accelerate rendering as shown in [Baboud et al., 2012]. In the following section we will focus on triangulated meshes and we will discuss several methods for generating triangular meshes from grid structures.

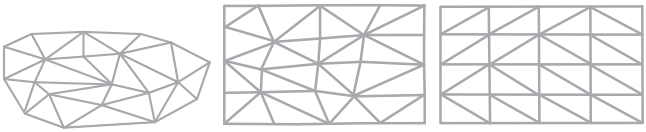
3.2.2 Terrain Triangulation

An overview over terrain triangulation and terrain rendering can be found in [Pajarola and Gobbetti, 2007]. Triangulated terrains can be divided into triangulated irregular meshes, also called triangular irregular networks (TIN), semi regular grids and regular grids as illustrated in Fig. 3.3. Here, we assume that a *DEM* is always interpreted as regular grid of data points. Therefore, for this work we will not go further into details about triangulated irregular networks or triangulation of semi regular grids, even if some algorithms or challenges might be applicable to these cases as well.

The triangulation of regular grids and regular height fields is discussed in detail in [Pajarola and Gobbetti, 2007] and other sources. For this reason, this section will focus on explaining major requirements to understand the challenges with regular grid triangulation. These requirements for terrain triangulation algorithms are listed in Tab. 3.1. In the following paragraphs, we will discuss these criteria in more detail.

Restricted vs. Non-Restricted Quadtree

Triangulation algorithms for terrains rely on hierarchical data structures to enable level of detail rendering. Usually, the *DEM* is processed beforehand to a hierarchical data structure such as a quadtree. A typical example is illustrated in Fig. 3.4. Details to this data structure can be found in the book of Hanan Samet [Samet, 2005]. A quadtree offers several advantages such as out-of-core data loading, faster spatial queries on the terrain information and in some cases data compression. The triangulation of the terrain mesh has to be adapted to such quadtree data structure. There are two categories of quadtrees, namely non-restricted quadtrees and restricted quadtrees. The difference is that a restricted quadtree assumes for every node that all it's neighboring nodes have not more than 1 level difference to



(a) TIN (b) Semi Regular Grid (c) Regular Grid

Figure 3.3: *(a) A triangulated irregular network. (b) A semi regular grid with arbitrary movements of equally distributed points. Grids are also called semi regular if the distances between point rows or columns differ. (c) Example for a regular grid with an arbitrary triangulation. In regular grids, data points have the same distance to it's neighbours within all rows and within all columns.*

this node. It holds that every node v in a quadtree $Q = (K,E)$ has a depth $d(k)$ which is the shortest path distance to the root node within the tree Q and has a set of neighboring nodes $S = \{k'_0, ...k'_n\}$, $S \in Q$. In a restricted quadtree the following holds for every node $k \in K$:

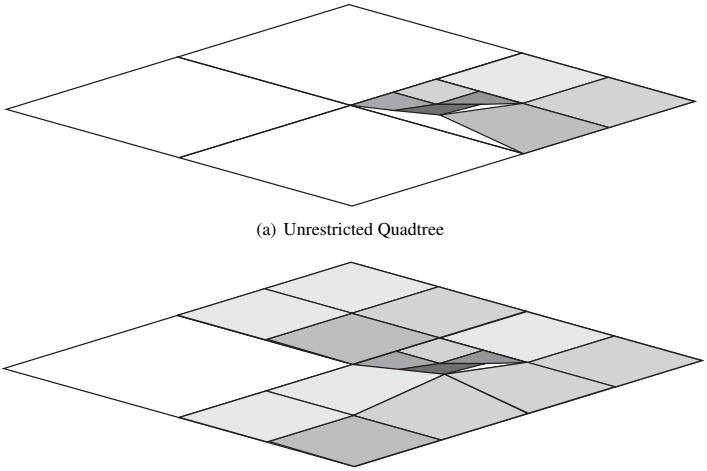
$$|d(k) - d(k')| <= 1 \; \forall k' \in S \tag{3.1}$$

Fig. 3.4(a) shows a non-restricted quadtree whereas a restricted quadtree can be seen in Fig. 3.4(b). In both illustrations we can see errors in the mesh caused by differing height values at T-joints. These errors are called cracks. Cracks can appear in quadtree meshes, whenever an edge has less vertices than a neighboring edge and therefore, a differing height function at these vertices.

Obviously these errors only appear if the height value at a point differs from the height values of the neighboring vertices. Such cracks and T-joints are the main reason why we distinguish between continuous and non continuous triangulations.

Continuous and Non-Continuous Triangulations

Example for continuous and non-continuous triangulations are shown in Fig. 3.5. A continuous triangulation simply means that all T-joints and cracks in the mesh structure are eliminated. A continuous triangulation is often required for several reasons. The main reason is the visual disturbance of cracks in the mesh. But also, because more advanced mesh processing algorithms could rely on the requirement of a correctly connected mesh. This might not be important for purpose of visualization of height values but it can get important for subsequent algorithms such as fluid simulations, collision detection on the terrain or interactive terrain deforma-



(a) Unrestricted Quadtree

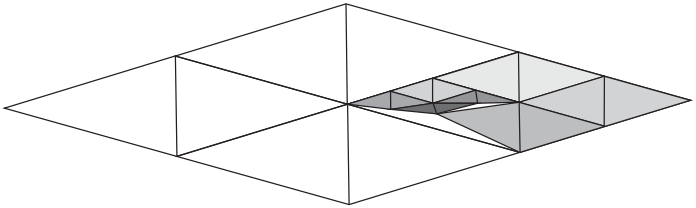
(b) Restricted Quadtree

Figure 3.4: *(a) A non-restricted quadtree containing cracks on multiple positions. (b) A restricted quadtree containing a crack at a T-joint.*

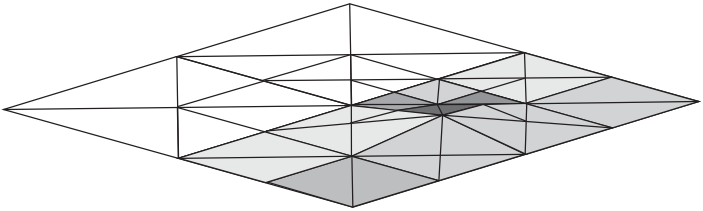
tion and editing. Algorithms or systems using a non continuous triangulation often use so called skirts for filling the cracks. This technique is discussed in [Thatcher, 2002]. Skirts are extensions to the triangulated mesh, such as ribbons around the quad tree tiles with the purpose to hide the height difference at T-joint locations. However, introducing skirts can lead to texturing artifacts by itself and the whole terrain mesh topology will still be unconnected. Furthermore, skirts produce artificial data within a visualization which should be avoided if possible.

Triangle Soups, Strips and Fans

Earlier triangulation research in computer graphics often had a focus on generating triangle strips and fans from a triangle soups. Graphics cards offer the functionality to interpret triangles in different ways. A triangle soup can be defined as a group of individual not connected triangles in space as shown in Fig. 3.6(a). This representation assumes that every triangle is defined with three vertices. But there are two other major ways to describe triangles by reusing vertices of a preceding



(a) Non continuous Triangulation

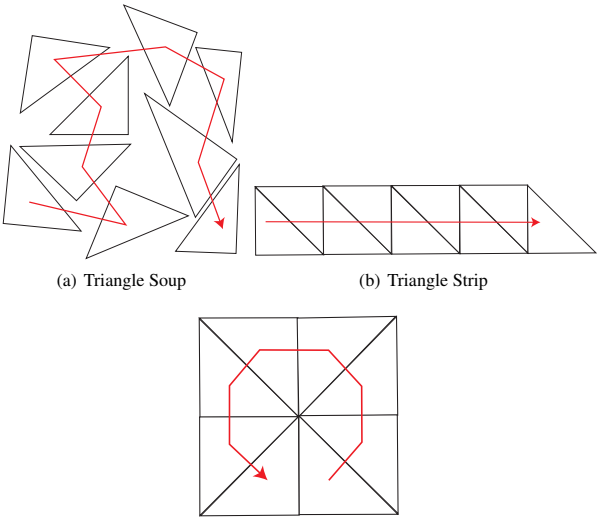


(b) Continuous Triangulation with RASTeR

Figure 3.5: *(a) A none continuous triangulation example containing multiple cracks and T-joints. (b) A continuous triangulation example how the RASTeR algorithm produces it.*

triangle, specifically triangle strips and triangle fans. An example for a triangle strip and a triangle fan is illustrated in Fig. 3.6(b) and Fig. 3.6(c). A triangle strip starts with an initial triangle and defines every subsequent triangle by two preceding vertices from the last triangle and a new vertex. Fig. 3.7 shows the appearance of vertices in detail. A triangle fan is defined with the first vertex from the first triangle, the preceding vertex from the last triangle and a new vertex.

Fig. 3.7 illustrates how the triangle arrays are structured for the graphics card. A simple triangle soup is defined as an array of vertices. This array is interpreted in such a way that groups of three consecutive vertices form a triangle. Thus, vertices are repeated for connected triangles. Triangle strips and triangle fans have the advantage that from the second vertex on every vertex defines a new triangle. Connected triangles can be stored more effectively with triangle strips and fans. This improvement can be particularly advantageous for older graphics hardware since a reduced amount of triangle information can lead to a performance gain during rendering. However, due to the parallel setup in modern graphics hardware, the sequential ordering of triangles and vertices has lost priority in comparison



(a) Triangle Soup (b) Triangle Strip

(c) Triangle Fan

Figure 3.6: *Examples for a triangle soup (a), a triangle strips (b) and a triangle fan (c).*

to other issues, e.g. the number of rendering calls. The reason for this is that vertex information can be reduced efficiently by using index structures as well. Methods using these different primitive types specifically for terrain triangulation can be found in [Pajarola, 1998], [Pérez et al., 2004], [Schneider and Westermann, 2006], [Lim and Choi, 2008] or [Pajarola and Gobbetti, 2007] and many other sources. Furthermore, Sec. 3.3.1 describes how triangle strip generation can be implemented with the *RASTeR* algorithm.

Indexed and Instanced Vertex Structures

The idea behind an indexed vertex structure is that the triangulation information is not expressed as consecutive vertices in an array, but rather as consecutive index values referencing to entries within the vertex array. An illustration for this is given in Fig. 3.8. The advantage of such a setup is that the vertices of the vertex array do not require a strict ordering according to the triangle structures. Another advantage is that redundant vertices do not appear multiple times within the vertex

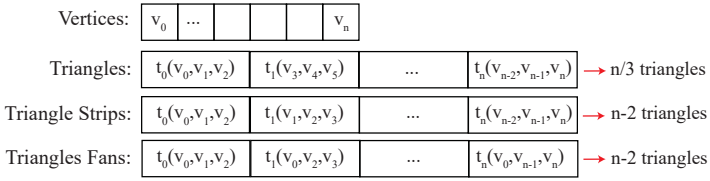


Figure 3.7: *Triangle strips and fans can be used to reduce the amount of vertex information or index information per triangle by providing vertices with a specific ordering inside an array. The amount of representable triangles increases from $n/3$ to $n-2$.*

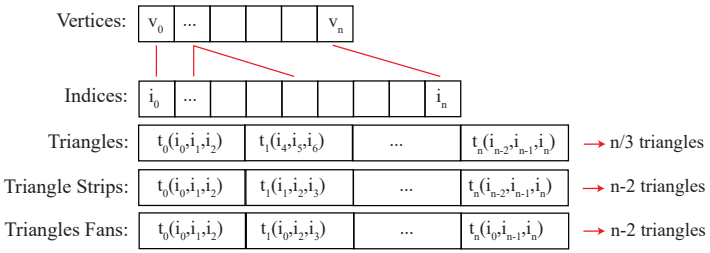


Figure 3.8: *Triangle strips and fans can also be used together with vertex indices so that the ordering of vertices can be expressed in an index array.*

array for triangles sharing edges. Therefore, *DEM* grid structures with several million triangles profit from indexed vertex information because of the reduced vertex-related memory operations on CPU and GPU.

Another recently introduced concept in GPUs is instanced rendering. Modern GPUs work with so called triangle patches. A triangle patch is a set of triangles, such as the ones in Fig. 3.6. With instanced rendering, the GPU is able to render a patch multiple times with a single draw call by reusing the vertex information of a single patch. The advantage of instance rendering is its flexibility to define vertex attributes per patch or per vertex. Therefore object instances can slightly differ in appearance while most of the vertex information is shared from a single instance.

3.2.3 Terrain Triangulation Algorithms

Many algorithms exist to provide continuous triangulations for terrains. References can be found in the survey [Pajarola and Gobbetti, 2007]. The already mentioned *RASTeR* system described in [Bösch et al., 2009] and [Goswami et al., 2010] can be seen as an example of a bintree triangulation and will be described in the following sections in more detail.

The clipmap approaches described in [Clasen and Hege, 2006], [Lambers and Kolb, 2012] and [Dimitrijević and Rančić, 2015] are using a view frustum based triangulation method and can provide a continuous triangulation as well. For a spherical clipmap however, there is the problem that the clipmap vertices are not correctly matching with the *DEM* height values. This leads to rendering errors because of imprecise calculation operations, such as height values in the visualization, that do not exist in reality but are results of filtering several neighboring height values.

An example for a commercial product is the triangulation system in the *Frostbite* engine described in [Andersson, 2007]. In comparison to our system, the *Frostbite* engine triangulates the quadtree tiles directly. As well as our approach, it uses a predefined set of patches to render the triangle mesh. In comparison to the *RASTeR* algorithm, this produces another type of continuous triangulation where the level of detail in the triangulation is not as smooth as it can be in *RASTeR*. However, it is easier to implement since there is only a quadtree to take care of.

Recent research in terrain rendering often focus on triangulation with hardware tessellation, e.g. [Ripolles et al., 2012], or solve the triangulation of the whole virtual globe sphere directly on hardware, as in [Kooima et al., 2009]. The main focus lies then on efficient geometry compression methods, as shown in [Bettio et al., 2007] and [Dick et al., 2009b], or on efficiently stored structures for instanced rendering, as in [Livny et al., 2009].

3.2.4 Textures for Terrain Rendering

An important aspect for terrain rendering systems is texture handling. Texture based requirements have more and more influenced terrain rendering systems. Textures are the main source of memory requirements nowadays in terrain visualization. Thus, texture compression has to be used, such as the standard compression techniques available on GPUs nowadays[2]. However, there are terrain rendering specific compression approaches too. An interesting system using a GPU encoding and decoding pattern for height field and texture data can be found in [Treib et al., 2012]. Another article only focusing on the compression of *DEMs*

[2] OpenGL Extension: GL_ARB_texture_compression

can be found in [Durdevic and Tartalja, 2013]. Its main idea is to use Bézier surfaces to approximate the height field values. Another recent trend for large-scale texturing is virtual texturing on graphics devices[3]. For this work, texture compression will not be discussed further since it is possible to use standard compression techniques with *RASTeR* for texture handling.

As shown in Tab. 3.1, some triangulation algorithms require a specific texture size. Such algorithms assume that vertices on quadtree borders overlap with other vertices. This overlap guarantees that these vertices have the same height position. Therefore, the preprocessing of the textures has to be adjusted to these requirements.

Another aspect of textures is the rendering pipeline setup. Often, geometric primitives are ordered according to the appearance of their textures to avoid expensive texture switches during rendering. Usually, textures are bound to a specific texture layer which can be enabled or disabled during rendering. A shader program can only access textures from enabled texture layers. Recent graphics hardware offers the possibility for so called resident or bindless textures. When a texture is declared as resident it is possible to access the direct memory pointer to the texture memory on the GPU and use this information as a per vertex information in the shader program. Our terrain rendering system explained in 3.3.1 can optimize the terrain rendering process significantly if this GPU extension [4] is available to the hardware where the terrain visualization is executed.

3.2.5 Terrain Web Services and Virtual Globe System Requirements

Past research in terrain rendering often assumed that the terrain information can be simply a single *DEM* file. With more demanding user access requirements and growth of terrain data, it is required to introduce client server architectures to access large-scale data over terrain web services, so called *web map services* (*WMS*).

Depending on the application, the server side provides either direct access to terrain information, similar to a database, or any kind of preprocessed format or folder structure populated with several thousands of *DEM* files for fast access to the terrain information. If a folder structure is provided, the *DEM* files are called tiles and represent nodes of a quadtree. Therefore, these web services are also called *Tile Map Services* (*TMS*). The system described in this work always assumes such a *TMS* to access the terrain information locally or remotely. Further details about tile-based systems can be found in [Sample and Ioup, 2010]. Vari-

[3] OpenGL Extension: GL_ARB_sparse_texture

[4] GL_ARB_bindless_texture

ous system architectures and more details about such system implementations are described in [Hildebrandt and Döllner, 2010].

Based on the requirements in Tab. 3.1, terrain visualization algorithms require specific preprocessing of *DEM* data to apply specific tiling patterns, compression algorithms, or formats. Therefore, storage systems are tightly coupled to the client software and not necessarily reusable for other clients without preprocessing the *TMS* structure again. Web service interfaces want to provide their data to a variety of clients, e.g. clients running 3*D* or 2*D* visualizations. Thus, open databases can not provide terrain or image data with support for *TMS* structures specific to certain algorithms. For this reason, terrain visualization systems often stick to simple terrain visualization algorithms.

WMS and *TMS* nowadays provide access to a huge amount of image information from different sources. It remains a challenge for terrain visualization to combine multiple *TMS* layers and additional forms of data, such as climate or traffic data, to one single visualization enriched from different sources.

Typical client-server systems are virtual globes. But virtual globe systems have an additional requirement to terrain visualization, namely the recalculation of the triangle structure to a spherical or ellipsoidal coordinate system. Tab. 3.1 shows which algorithms need adjustment when used in a virtual globe setting.

Other system requirements related to the choice of a terrain visualization technique are shown in Tab. 3.1, namely specific GPU requirements, expected implementation complexity of the system and the capability of interactive editing functions. In the remaining chapter we will show how to adapt the *RASTeR* algorithm to match these requirements.

	Criteria	RASTeR	Frostbite	Plain Quadtree & skirts	Geometry Clipmaps	Terrain Ray Casting
Triangulation	Quadtree	restricted	restricted	non-restricted	non-restricted	non-restricted
	Level of Detail	continuous KPatch triangulation	continuous quadtree tesselation	no continuous triangulation	discreet levels	no triangulation
	Triangle Strips	possible				
	Indexed triangles	possible				
	Instanced structures	possible				
Textures	Out-of-Core System	possible				
	Texture size Restriction	n^2+1	n^2	-	-	-
	Texture compression	no restrictions				
System	Rendering errors	none	none	cracks appear or distorted skirts	problems with high-frequency & high-amplitudes	distorted skirts
	Preprocessing	tiling DEM to quadtree structure				
	Implementation Complexity	bintree & quadtree	quadtree	quadtree	clipmap & quadtree	ray casting & quadtree
	GPU Requirements	profits from adv. features	profits from adv. features	none	none	fast GPU
	Interactive Editing	edit quadtree tiles				
	Multiple DEM & Imagery	blending possible				possible but slow
	Applicable to Spherical Rendering	possible			visualize and edit interpolated heights	spherical ray casting

Table 3.1: *The table summarizes the differences between selected previous methods and the updated RASTeR technique.*

3.3 The RASTeR System

As mentioned before, our *RASTeR* system is based on the former works [Bösch et al., 2009] and [Goswami et al., 2010]. However, our system is a complete new implementation of the *RASTeR* algorithm to show the applicability to the changes in user and software requirements, as presented in the sections before. In this section, we will first describe the concrete triangulation algorithm, then discuss dynamic level of detail functionality, and last, we will discuss the rendering pipeline including effects for terrain visualization.

3.3.1 Continuous Triangulation with RASTeR

In principle, *RASTeR* has two main components interacting with each other, namely a quadtree, also called *MBlock* quadtree, and a triangle bintree, also called *KPatch* bintree, holding the triangulation information. Examples for these trees are given in Figs. 3.9 and 3.10. In the next sections, we will discuss specific points for the quadtree and afterwards, we will explain how the bintree is used to achieve a continuous triangulation.

MBlock Quadtree

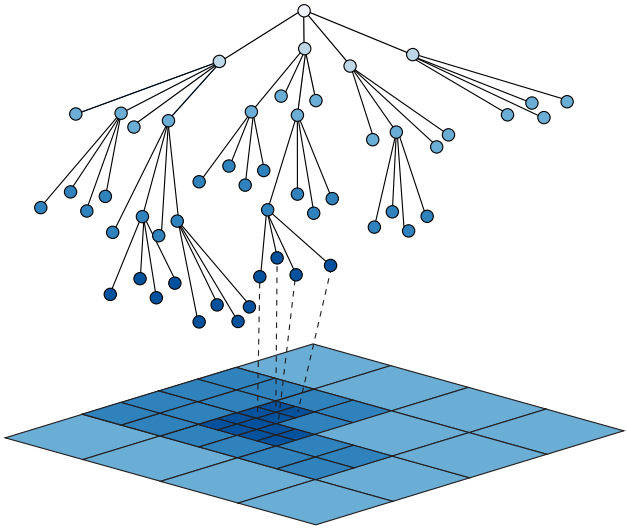
The *MBlock* quadtree is a traditional quadtree consisting of nodes which we call MBlocks. An MBlock quadtree is always restricted as defined in Sec. 3.2.3. A typical situation during rendering is shown in Fig. 3.9. The main purpose of the quadtree is controlling the tile based data interaction during runtime with the TMS.

Every quadtree node has a unique location inside the tree, consisting of three properties, namely x and y coordinate as well as the *LoD* level which equals the depth of the node in the tree. We call the tuple (x, y, lod) a $SpatialKey$.

These spatial keys are equivalent to the address inside the TMS. This is possible because of the assumption that the quadtree and the TMS folder structure exist in the same geographically referenced coordinate system. In our case, we always assume the $WGS84$ coordinate system[5]. When using this coordinate system, our visualization has to use two quadtrees to match an earth like ellipsoidal shape. One for each half sphere from -180 to 0 longitude and from 0 to 180 longitude. Details can be found in [Cozzi and Ring, 2011].

Due to the association to a geographically referenced coordinate system, the *RASTeR* system can request new nodes for the quadtree simply based on their spatial key information. Furthermore, it is possible to access the same spatial key in different TMS repositories, because the same spatial location is equal in

[5] http://spatialreference.org/ref/epsg/wgs-84/



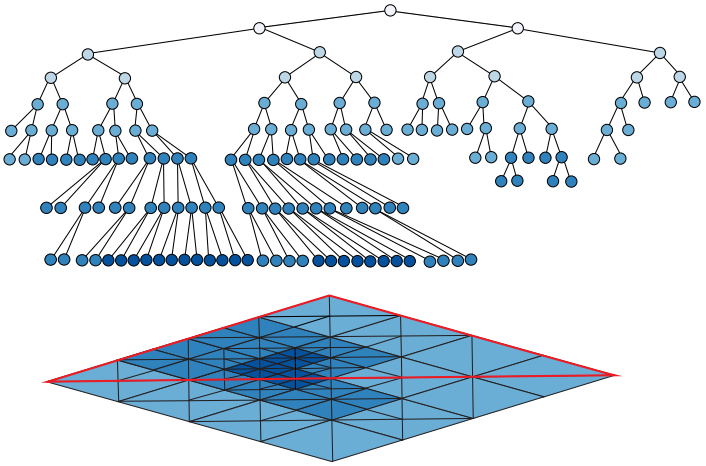
(a) MBlock Quadtree

Figure 3.9: *An illustration of a typical situation of a quadtree state.*

all TMS repositories. Spatial keys can also be used to store information about the lifetime of a key, e.g if the key is already available, requested, deleted or not available at all for a specific TMS layer. In our implementation, TMS keys are usually requested on the following events:

- The quadtree's restriction criteria changed.
- The *KPatch* bintree restriction criteria changed.
- The camera view frustum changed.

Typically, the loading of new data tiles from disk takes more time than rendering a single frame. Thus, loading tiles should be done in a multithreaded setup. In such multithreaded environments, the interaction of *KPatch* bintree and quadtree can be challenging and complex to implement. However, as long as the quadtree is guaranteed to be restricted, the triangulation of the *KPatch* bintree can be performed as explained in the following section.



(a) KPatch Bintree

Figure 3.10: *An illustration of the bintrees used to triangulate the quadtree shown in Fig. 3.9. The red section illustrates one of two KPatch bintrees and shows which nodes are covered with the KPatch bintree visualized above.*

KPatch Bintree

The *RASTeR* algorithm is based on the triangulation of a binary tree, also called bintree or *KPatch* bintree. A typical *KPatch* bintree triangulation is shown in Fig. 3.10. The illustration of the tree in Fig. 3.10 shows the state of the binary tree from the root node, marked by the red triangle. A binary tree node is called *KPatch* and represents a triangular region inside the binary tree. As visible in the image, the triangulation of a quadtree can be done with two *KPatch* binary trees. In opposite to a regular binary tree, the *KPatch* bintree has some specialties, namely the splitting behavior and the restriction criteria.

The splitting of a *KPatch* bintree follows a top down splitting order shown in Fig. 3.11. As illustrated, a *KPatch* can only be split into two specific other *KPatch* configurations. An overview over the specific split configurations can be seen in Fig. 3.12. From the figure it is possible to observe the formation of recursive patterns by the split configurations. Furthermore, it shows that the tile size for a quadtree node has to be $2^n + 1$ because the binary tree splitting will produce

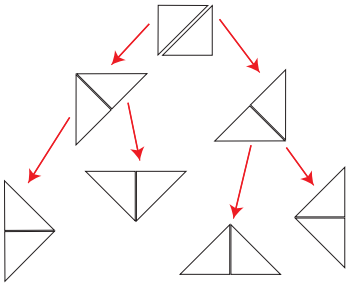


Figure 3.11: *An illustration for the KPatch bintree splitting behaviour showing the first few splits.*

always an edge length of $2^n + 1$ vertices for a complete tree. A *KPatch* split is performed under the following conditions:

- A *KPatch* is nearer to the camera in comparison to a predefined distance.
- A quadtree tile is available for the new *KPatch* level.
- The restriction criteria on the bintree is not violated.

The second characteristic is the restriction criteria for the bintree to avoid cracks and T-joints. Similar to a restricted quadtree, a *KPatch* bintree is restricted so that every node has only one level difference to it's neighbors. In earlier versions of *RASTeR*, this was achieved by using saturated error metrics based on the original data set, such as the one described in [Gerstner, 2003].

Using a fixed data based error metric has several drawbacks. First, the system assumes a preprocessed bintree error metric for each *TMS*, provided when accessing the *TMS* the first time. Second, the usage of a preprocessed error metric makes the possibility to combine multiple height fields much more complicated, because different data sets would have different error metrics. Third, for terrain editing operations, the error metric has to keep track of changes in the height field.

The forth drawback is that the saturated error metric utilizes the difference of saturated errors for different *KPatch* levels. The advantage is that the *KPatch* appearance inside an *MBlock* can be controlled more in detail. However, this advantage degrades when a higher tessellated *KPatch* size is used. This is, because a *KPatch* covers more data points and the chances are higher that the saturated errors of different *KPatch* levels converge. In other words, the chance for appearing height differences in a *KPatch* is higher when the *KPatch* size is bigger

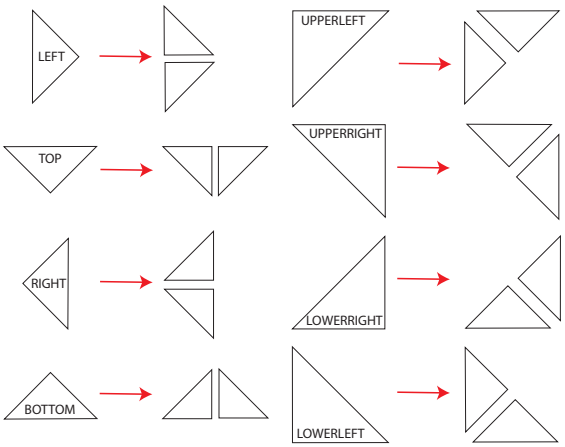


Figure 3.12: *All possible splits for specific KPatch configurations.*

since more data points are taken into account. Thus, high saturated errors appear in the deep levels of the tree. Most of the time, it will be required to split to the maximum depth anyway. On the positive side, a data based error metric can avoid unnecessary splits for some *KPatches* and therefore, reduces the triangle count.

However, in our new *RASTeR* system we decided to work without a data-based error metric to stay conform with the given *TMS* structure. To stay competitive in performance, we try to keep the amount of bintree nodes as low as possible, but raise the *KPatch* size, so that we have less calls of *OpenGL* draw functions but more triangles per function call. To render a *KPatch* we use a specific tessellation pattern.

KPatch Tessellation

After all visible nodes of the *KPatch* bintree are evaluated, we render the *KPatches*. An advantage of the *KPatch* configurations in Fig. 3.12 is that the tessellation pattern is the same for all configurations. We use the tessellation pattern in Fig. 3.13, because it is an easy way to express *KPatches* as one single triangle strip, using hidden triangles at the turns. In the illustration, we use a *KPatch* with 5 vertices per edge. In our implementation we use typically 33 or 65 vertices per edge, depending on the system capacities.

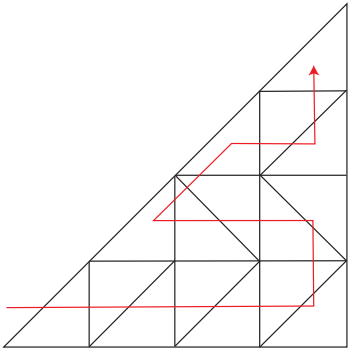


Figure 3.13: *An example KPatch showing the triangulation pattern for triangle strip generation.*

Furthermore, we use index arrays to describe the ordering of a *KPatch* triangle strip, as explained in Sec. 3.2.2. Since all *KPatch* configurations require the same amount of indices, it is possible to further optimize the rendering calls using instanced rendering. In the final version the implementation never sends a vertex position to the graphics card, because the position of a single index can be calculated by the coordinates inside the *KPatch*, the position of a *KPatch* inside the *MBlock*, and the position of the *MBlock* inside the global *MBlock* quadtree. This calculation is done in the vertex shader stage.

The system overview in Fig. 3.14 shows the process and the contents of the vertex buffers during rendering. It is worth mentioning that the texture coordinates are generated using the same information. The texture coordinates are needed to access resident textures over the vertex and fragment shaders. The resident texture handles are pointers to texture memory, set for every instance of a *KPatch*. The texture coordinate for a certain vertex is valid for all texture layers, because all image layers are in the same geographically referenced coordinate system.

For virtual globe applications, this tessellation pattern is usable to render the terrain on spheres or ellipsoids as well. To achieve this, the final vertex position can be converted into spherical coordinates within the vertex shader. In the following section we discuss shading effects implemented on top of our *RASTeR* system to improve the visual quality of the terrain visualization.

3.4 Results

During the advancement of this project, the new *RASTeR* implementation was enhanced several times. Furthermore, several shading effects and additional topics were implemented to improve the rendering quality and as showcases for what is possible to achieve in a terrain render engine. An example for the triangulation explained before can be seen in Fig. 3.15. In this section, we discuss details about the usage of normal vectors, multitexturing with alpha blending, ambient occlusion, edge highlighting and real-time terrain analysis.

Usually, rendering terrain is done based on an elevation color table, as shown in Fig. 3.17. It is possible to interactively change the color table and therefore edit the terrain color. The final image color is calculated by a combination of ambient and diffuse light using the normal vectors calculated by the elevation data.

In early stages of our new *RASTeR* system, normal vectors were generated online in the shader, based on a weighting of 4 or 8 neighboring height values. Resulting images for this can be seen in Figs. 3.18, 3.19 and 3.22. However, this can be costly in terms of performance. Therefore, the future direction for the system will be to use preprocessed normals.

This decision has a performance tradeoff, because the normal vectors have to be loaded from a *TMS* as well. Preprocessing the terrain normals has the advantage that normal calculations can be done on the whole data set beforehand. Thus, no border artifacts appear because only single terrain tiles are accessible. In the current implementation we provide a preprocessor application taking care of different normal calculations. In addition, we use some optimizations from [Munkberg et al., 2006] to compress 16-*bit* spherical normals. We plan to further enhance the normal processing to improve the performance.

More sophisticated methods for preprocessing are conceivable. For example, the assumption of an octagonal normal distribution instead of a spherical one, to achieve more evenly distributed normal vectors when using a compressed format.

Multitexturing with Alpha Blending

Multitexturing is a well known rendering technique in computer graphics. The idea is to combine textures on top of each other and use a blend function to show or animate the transition from one texture into the other. A typical example is the creation of a static light effect. The combination of a texture and a light map generates the impression of a light cone on the texture.

This technique enables the possibility to achieve various rendering effects without changing the texture content. Moreover, it is possible to reuse the individual textures for other purposes and therefore, reduce the memory costs. In our system, we can use multitexturing to fade between multiple texture layers as

seen in Fig. 3.20. But also other applications are imaginable such as editing functionality using a brush tool or light and shadow effects on the terrain for static light scenes.

Ambient occlusion

Ambient occlusion is a technique in computer graphics, to darken regions in an image with surfaces near tight corners or concave areas. With a standard light model, these surfaces receive the same amount of light as edges heavily exposed to the light source. However, tight corners or inward looking edges appear darker in reality, because surrounding objects also influence the lighting. Usually, this is not taken into account by standard light models.

Typically, the precise calculation of scene light using photorealistic rendering functions is too expensive for real-time rendering purposes. Thus, there are image based post-processing methods to approximate this effect, such as screen space ambient occlusion. Detail about the method can be found in [Bavoil and Sainz, 2008], [Shanmugam and Arikan, 2007] or [Hoberock and Jia, 2007]. Ambient occlusion requires the scene depth information to calculate the ambient occlusion map. Therefore, a multipass rendering process is required using a G-buffer providing the scene depth information. Fig. 3.16 shows the multipass terrain rendering setup in our system.

In our terrain rendering system the effect works well to emphasize valleys and steep mountain peaks. A comparison between traditional rendering without ambient occlusion and a rendered result with ambient occlusion is shown in Fig. 3.21. In the image, the valley and mountain cliffs are mildly more dark. The effect does not drastically change the image appearance. Hence, the user is not distracted by the effect.

Edge Highlighting

In perspective $3D$ scenes, the amount of elevation information can overwhelm users. In opposite to the real world, the user often sees the terrain from an unfamiliar perspective. Similar to static maps in $2D$, it is useful to introduce visual guiding for the user. One of this guiding effects is our terrain edge highlighting. The main idea is to highlight shapes of mountains and visually help the user understand the elevation data in a perspective view. We achieve edge highlights by using a *Laplace-Filter* for edge detection and colorization of the resulting pixels in a customizable color. Edge highlighting naturally helps the user to identify depth discontinuities. An example for this effect can be seen in Fig. 3.22.

Real-time terrain analysis

Interactive terrain analysis is a collection of rendering techniques to visualize terrain properties such as slope, aspect or flow. Further details to this properties can be found in the book about digital terrain modeling [Li et al., 2005]. In our implementation, we offer the possibility to interactively explore the terrain data appearing with different terrain properties, such as aspect, roughness, curvature, and multiple slope styles. An example for a slope view can be seen in Fig. 3.23.

Furthermore, we plan to introduce such terrain shading also for some hydrological properties such as the flow direction. However, possible scenarios are texture based flow simulations or more advanced fluid simulations for the visualization of landslides, water floodings or avalanches.

3.5 Conclusion

Terrain rendering is a very system oriented research field. A lot of functionality and knowledge has to be put together to make a terrain rendering system work. In addition, many requirements and system factors are influencing the development of a terrain visualization application. Furthermore industrial products got a lot of attention and raised expectations for requirements and innovations in systems. Even if systems developed by research institutions may have a limited set of features, it is still necessary to compare to industrial standards as well. A fair comparison of terrain algorithms is hard to achieve, because a lot of programming effort is needed and system-specific optimization techniques are available and might influence a comparison result significantly. However, the terrain rendering system of our *GlobeEngine* framework will be flexible enough to implement and compare our *RASTeR* algorithm with other terrain triangulation and raycasting algorithms, to make a fair comparison in visual quality, performance and robustness.

In this chapter, we showed a way to implement the *RASTeR* algorithms with another set of requirements using a tile based system such as a *TMS*. We explained how the *MBlock* quadtree and *KPatch* bintree have to work together to achieve a continuous triangulation. Furthermore it is shown how a *KPatch* bintree can triangulated to be able to achieve an error free rendering for spherical rendering on a globe. Furthermore we showed that the system is capable of handling shading effects for visualization and analysis tools as well.

A main limitation of the *RASTeR* system is the increasing complexity of implementation. To develop a binary tree for a continuous triangulation is additional effort in terms of implementation and maintenance work. The advantage is a slightly smoother triangulation in compare to a direct quadtree triangulation. For our group, the *RASTeR* system is a stable solution to build up other research projects as shown in the next chapters.

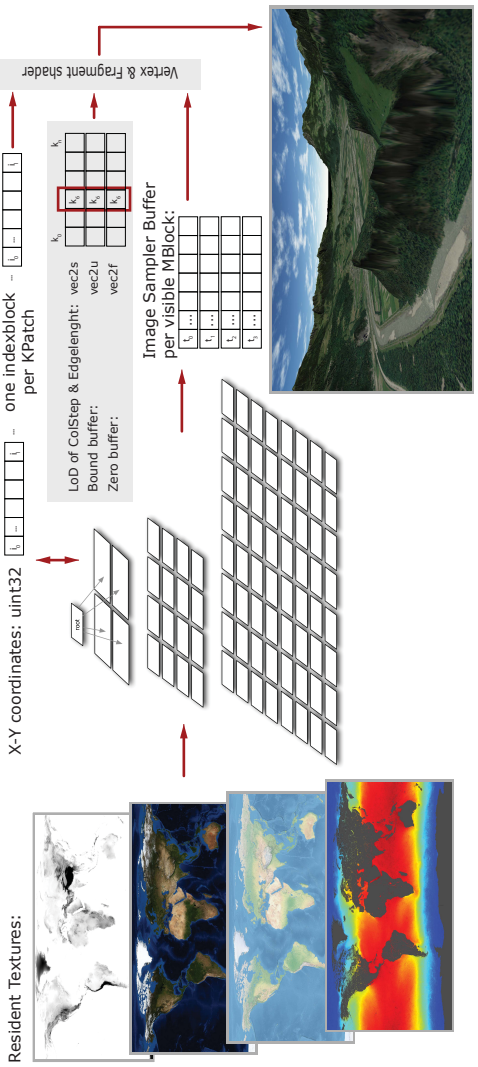


Figure 3.14: *An overview over the RASTeR Terrain rendering technique.*

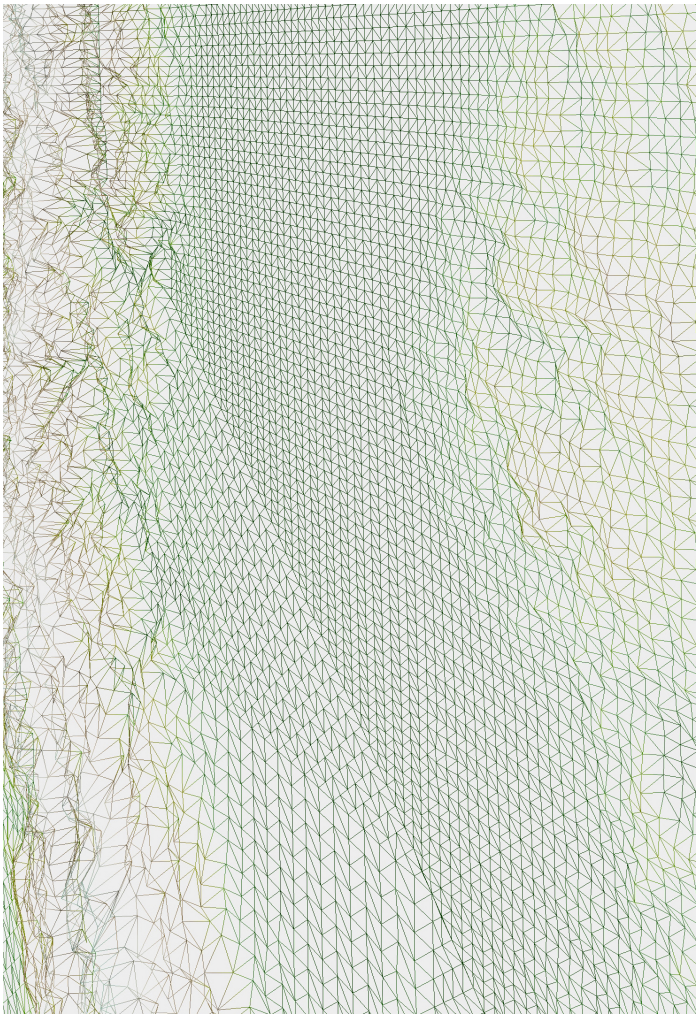


Figure 3.15: *Example scene showing the RASTeR triangulation using the World SRTM*1 *data set.*

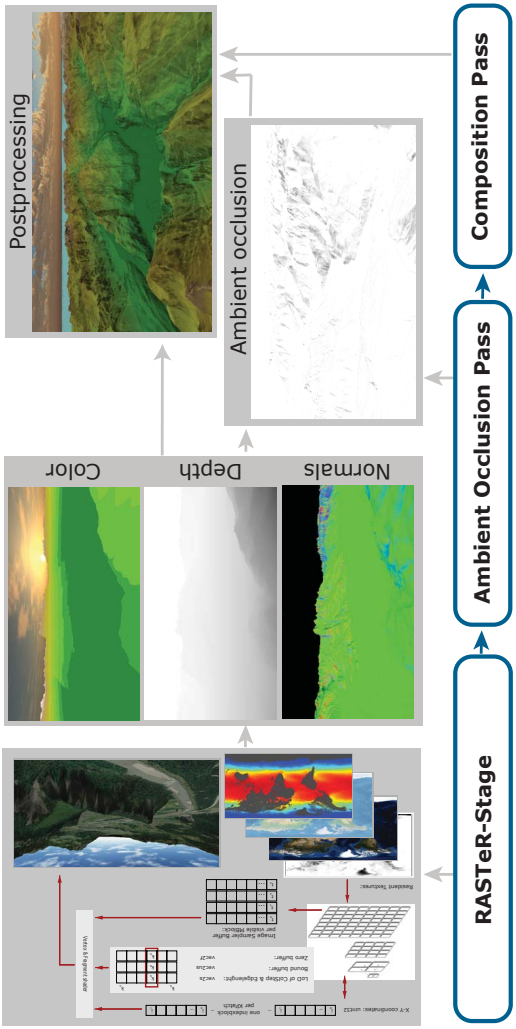


Figure 3.16: *An overvier over the terrain rendering pipeline showing the ambient occlusion pass in a multipass environment.*

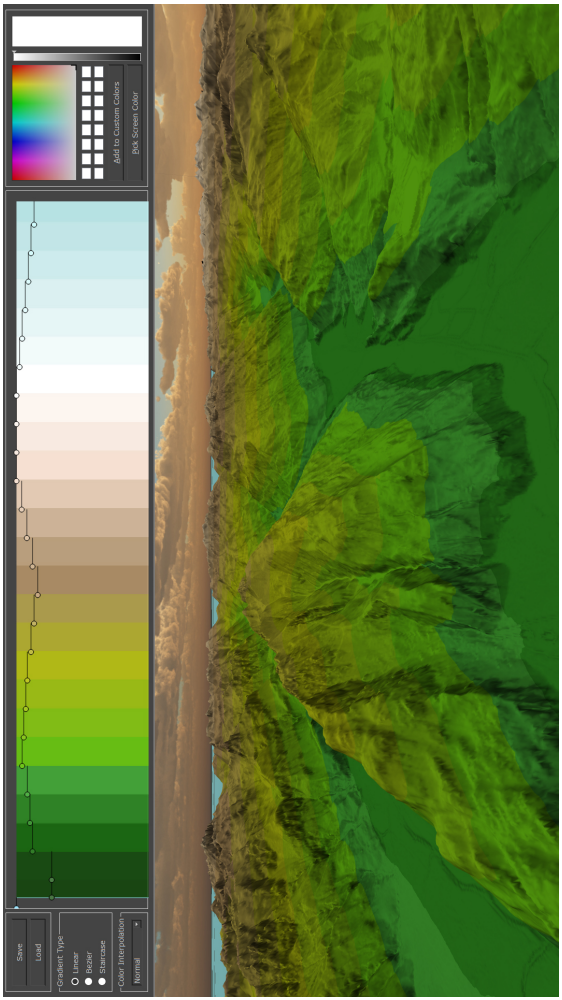


Figure 3.17: *Example scene showing the Vorarlberg data set with a height colorization pattern. The top of the image shows the transfer function editor for interactive editing the colorization pattern.*

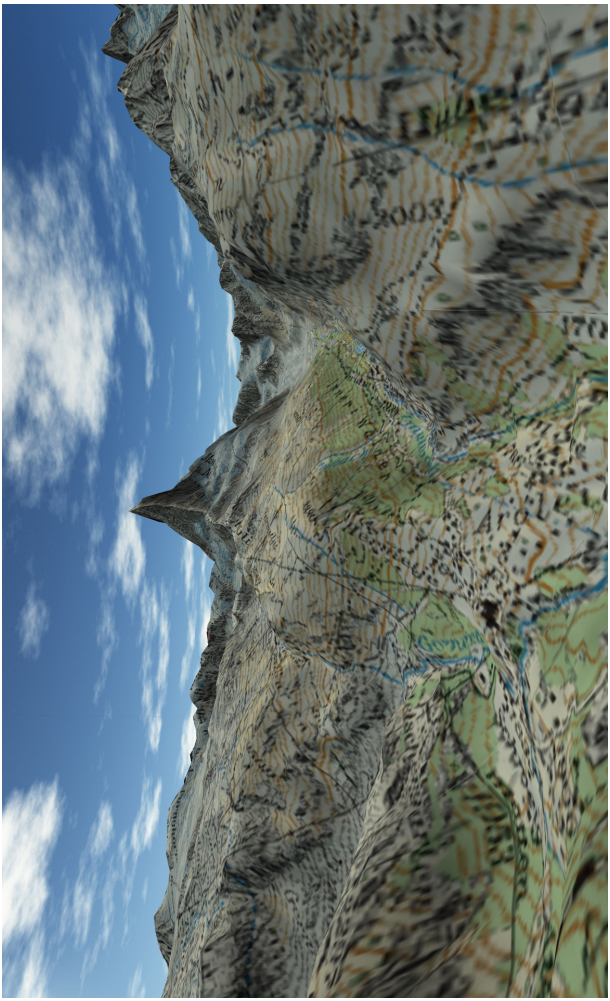


Figure 3.18: *Example scene of the swissALTI3D elevation data set showing the Matterhorn.*

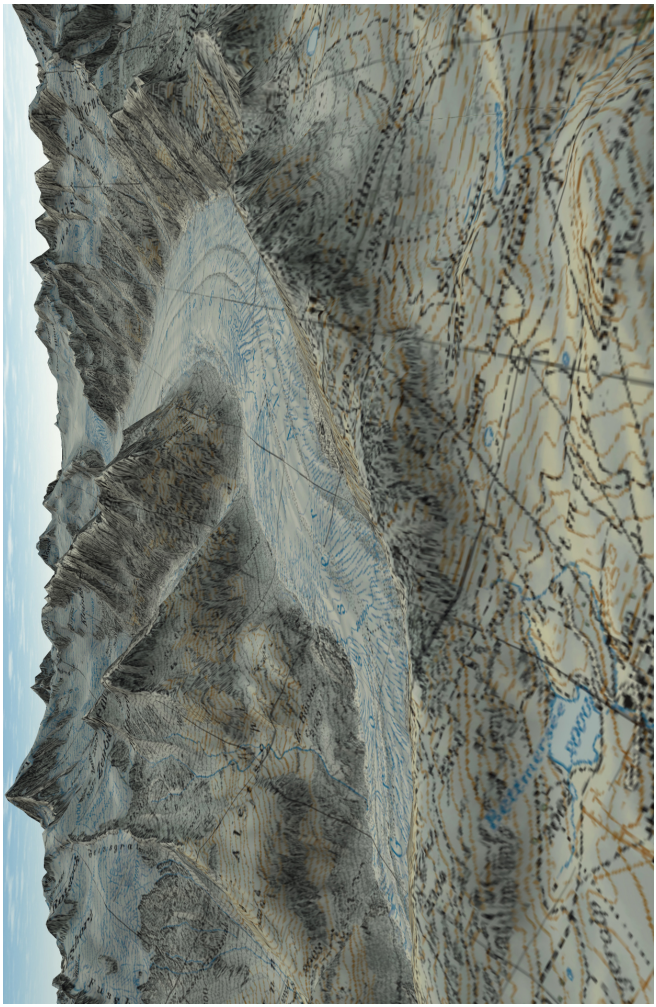


Figure 3.19: *Example scene showing the Aleetschgletscher using the swissALTI3D elevation data set with textures from the SwissMapRaster50 data set.*

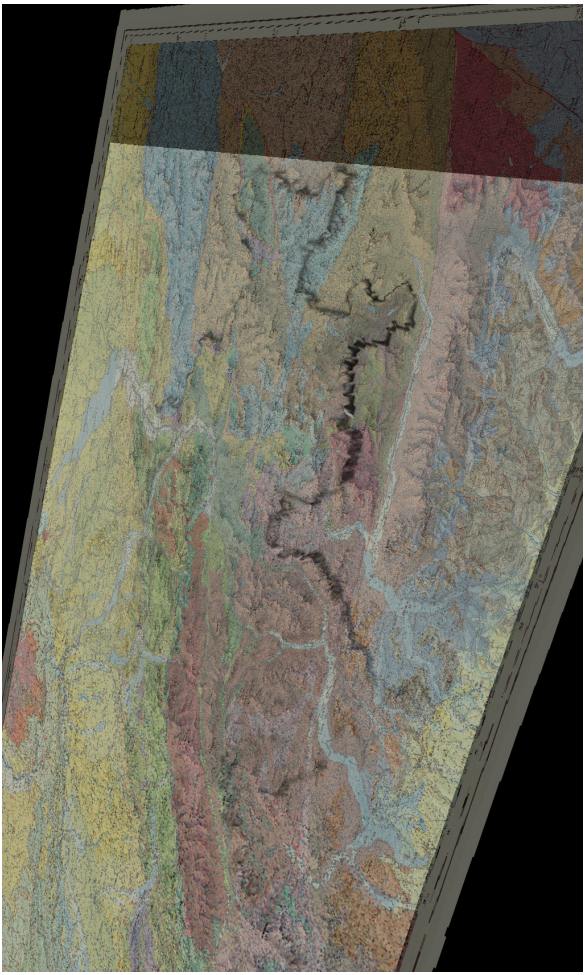


Figure 3.20: *Example for two alpha blended texture layers. The texture layers used in this screenshot are the geological map and the tectonic map of Switzerland.*

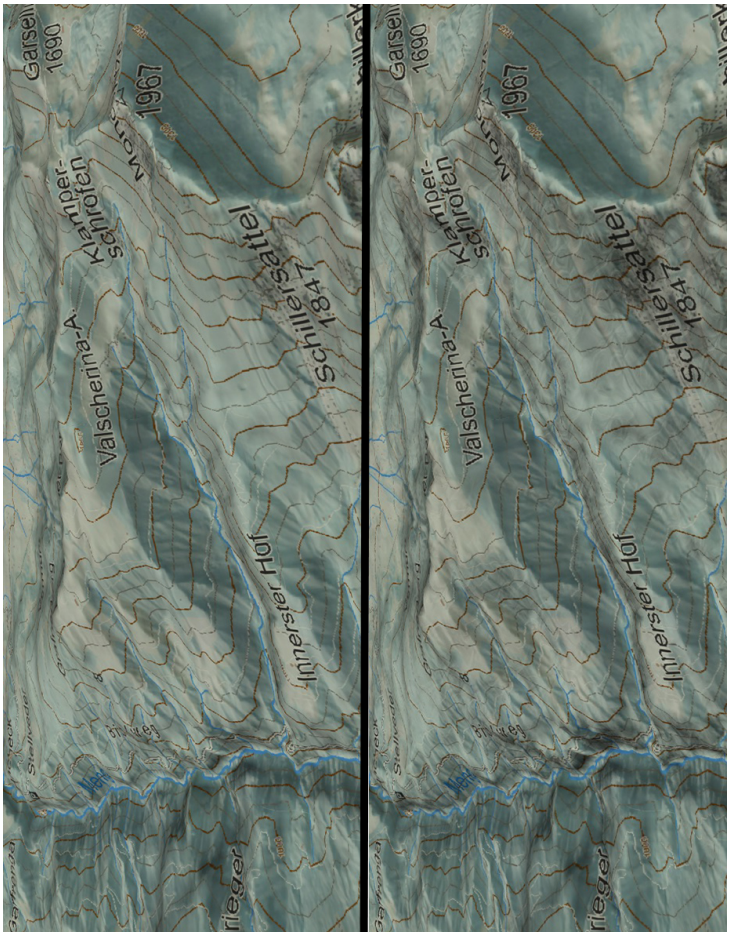


Figure 3.21: *Screenshot showing a part of the Vorarlberg elevation data to compare normal rendering (top) with active ambient occlusion rendering (bottom). Ambient occlusion causes a darkening of pixel which are near corners and edges to emphasize these regions.*

Figure 3.22: *Screenshot showing a part of the region around the city of Bad Ragaz using the swissALTI3D elevation data set and the SWISSIMAGE50 data set with active edge highlights in white.*

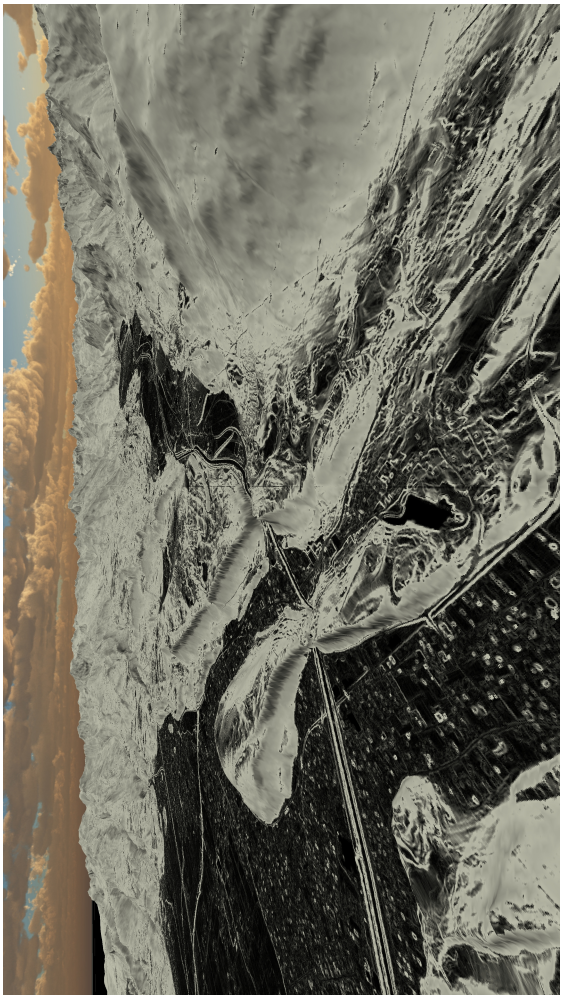


Figure 3.23: *An example screenshot for terrain analysis rendering. In this mode the terrain shows the values for slope.*

CHAPTER 4

RENDERING OF VECTOR INFORMATION

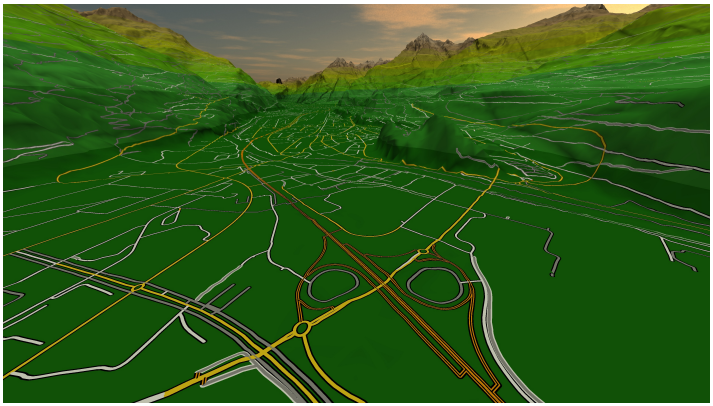


Figure 4.1: *Example vector map rendering of Lake Walensee.*

4.1 Point Features

The most simple form of features are point features. Point features can be defined as a set of three dimensional points $P = \{p_0 ... p_i\}$, where $p_i = \{x_i, y_i, z_i\}$. As seen in Sec. 2.1.3 a lot of GIS information is wrapped in point data sets and therefore associated with a spatial location. These data set can describe a simple location, such as a railway station, a volcano or a point of interest. Usually these data sets also contain additional data to this location which are much more interesting than the location itself such as voting results or economical data. In these cases the visualization pipeline for point data set has to be adapted and more meaningful visualization techniques have to be used such as charts. Fig. 4.2 shows the visualization pipeline for point visualizations in *GlobeEngine*.

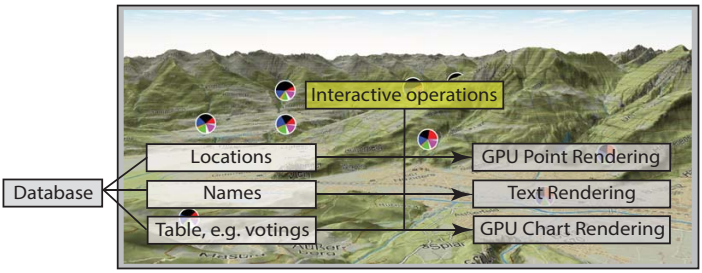


Figure 4.2: *The point visualization pipeline.*

In computer graphics, then rendering of point based data received a lot of attention, producing various techniques that can be used for rendering large amounts of point data. Unfortunately, these models do not specifically cover chart objects, so we have to adapt some of the techniques for large-scale chart information as well.

4.1.1 GPU Based Point Information

Point splatting is a well-known technique to visualize point based data sets. Point splatting can be used very efficient for point information as shown in [Gross and Pfister, 2007], [Kobbelt and Botsch, 2004], [Sainz et al., 2004]. The key idea is to display points by deforming each point shape in a way that no holes are visible in the resulting image. The technique indicates that there is no complex mesh structure generated and therefore, the main advantage of point splatting is

the reduced geometric complexity. This visualization technique fits perfectly the needs for our large-scale point datasets.

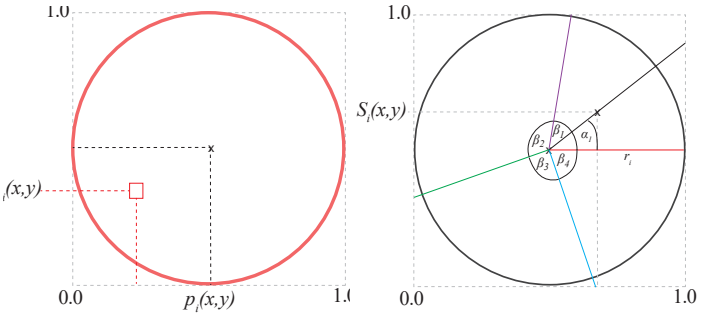
The original point splatting technique uses a textured rectangle for every visible point containing the pixel information of the point splat. The textured rectangle can be stretched or otherwise adjusted based on the camera's viewing angle to cover bigger or smaller regions with one single splat image. Modern *GPUs* take over this concept for their hardware accelerated point rendering pipelines. Fig. 4.3(a) shows the semantic structure of a colored point within the point-rendering pipeline.

When the point data runs through the rendering pipeline described in Sec. 2.1.4, the points get mapped from their $3D$ coordinate $p_i(x_i, y_i, z_i)$ to something what is called a fragment coordinate $s_i(x, y, d_{(x_i,x_j)})$. Often fragments are misinterpreted as pixels on the screen. But the major difference between a fragment and a screen pixel is that fragments are pixel coordinates per object. This means that for one single screen pixel $S(x, y)$ on the final image it is possible to have multiple fragments $s_i(x, y, d_{(x_i,x_j)})$ influencing the color of the final screen pixel $S(x, y)$. Often the most upfront fragment is used, i.e. the one with the smallest depth value, and all others are discarded. This is what we call depth culling or z-culling, because the depth values of the fragments are compared.

At this stage, the points are rendered as squared areas according to the given point size, as visible in Fig. 4.3(a). The point size specifies the point shape extent in screen pixel. A fragment shader program is performed for every fragment of this area. Within this fragment shader program, it is possible to access the fragment coordinate $s_i(x, y, d_{(x_i,x_j)})$ and also the fragment coordinate relative to the point shape. This fragment coordinate relative to the point shape is aligned between 0.0 and 1.0 within the point splat. Up to this step, the point shape is still a rectangle. The circle shape in point rendering appears, because the fragment program discards all fragments farther than a certain distance to the center point.

For our chart rendering, we used the hardware accelerated point splatting method by extending this fragment shader functionality. Fig. 4.4 shows results from our application using point splatting for charts. We implemented three types of charts: bar charts, pie charts, and rose diagrams.

- **Bar charts:** A bar chart is computed by the division of the x-axis according to the number of incoming data table columns. The fragment shader decides about the correct color and group for every fragment. Depending on the amount of columns n of the incoming data table, the bar sections can be identified by $x \mod n$ for every fragment. The fragments color can be chosen based on a color predefined in the data table. Fragments exceeding bar section are discarded to make fragments not belonging to bars opaque.
- **Pie charts:** Pie charts can be computed using a division of angle relations



(a) Point Splat (b) Pie Chart

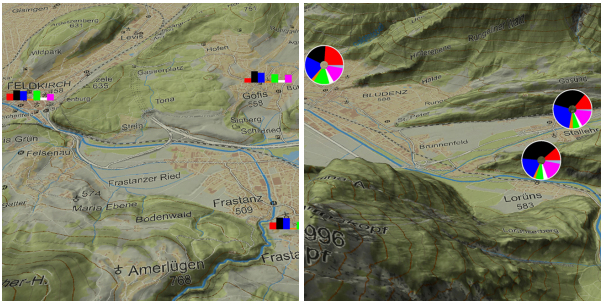
Figure 4.3: *Two case for the chart rendering.*

from the polar coordinates of a chart. Figure 4.3(b) shows a sketch of the concept. The resulting sections correspond to the data table columns. For every fragment $s_i(x, y, d_{(x_i,x_j)})$ within a point splat, the corresponding angle α_i is computed. This angle α_i is compared against every angle β_j. β_j are computed from the input data values and represent relative percentages to $\sum_{j=0}^{n}(\beta_j)$ for all j between 0 and the amount of table columns n. Analog to bar charts, the color value is chosen by the index j from a color lookup table. In addition, a fragment is discarded if the distance to the center is bigger than the splat radius r to produce a round shape.

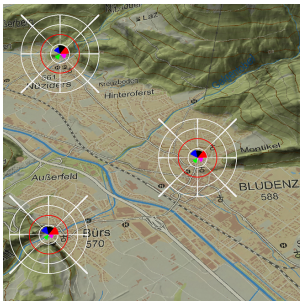
- **Rose diagrams:** Rose diagrams can be seen as combination of pie and bar charts. In case of a rose diagrams, pie sections have the same angle size, but each slice differs in extent. This means, that $x \mod n$ is done based on the angle, so that the section or group can be identified for every fragment $s_i(x, y, d_{(x_i,x_j)})$. In addition the radius of each section is adjusted to relative values.

4.1.2 Results

Results of our chart rendering can be seen in Figs. 4.4, 4.5 and 4.6 for two different voting data sets. Fig. 4.6 shows a voting dataset from 2012 containing 4588 charts. Our implementation runs on an Intel Core i7 3.5 GHz, 16 GB RAM, Nvidia GTX1080 (4GB RAM, 3840×2160). The data set can be explored interactively in $3D$. For the Vorarlberg charts as well as for the US voting data we



(a) Bar Chart (b) Pie Chart



(c) Rose Chart

Figure 4.4: *Chart Renderings of voting results for the Austrian National Council in 2013.*

achieve rendering times around 3-4 milliseconds per frame. The technique can also be applied on mobile devices or other portable graphic hardware with lower hardware capabilities.

Figs. 4.4 show the different kinds of charts described above on the basis of the national elections in Austria in the year 2013. The rose diagram example in Figure 4.4(c) shows a red line within the rose diagram describing the 50% border, which means for the voting example that the absolute majority is reached. These simple examples show that the technique is applicable to more complex chart structures. Furthermore, Fig. 4.5 shows that a full integration of such charts can be achieved within the perspective 3D view of a terrain render engine.

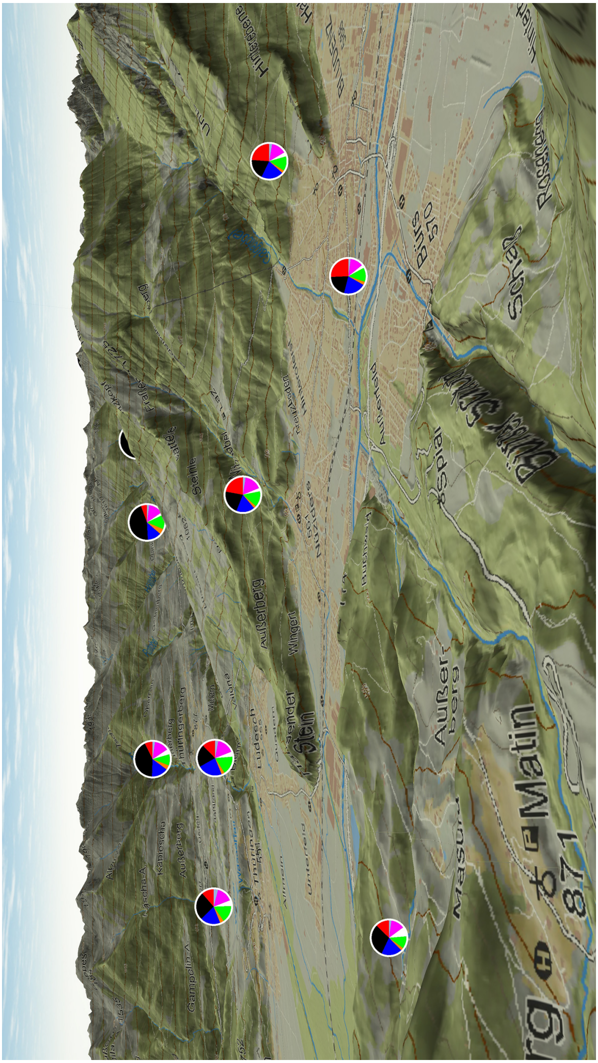


Figure 4.5: *Perspective rendering of voting results using pie charts.*

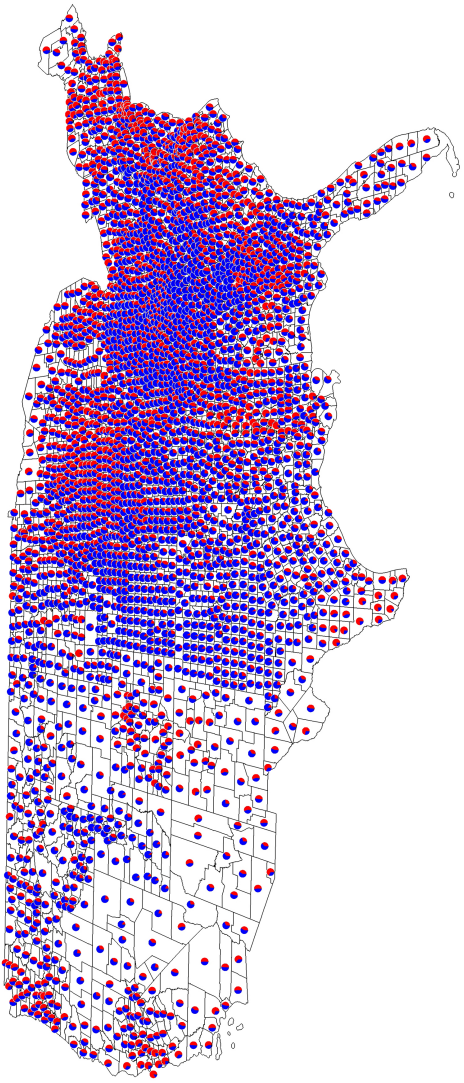


Figure 4.6: *Interactive GPU based chart rendering using the US Election 2012 data set.*

4.2 Line Features

Line features are a very essential part of vector map visualizations and virtual globe software systems. Improving the interactive visualization of vector maps will allow these software products to show and interact with more data and in a more precise way. Vector maps are used to represent geographic features such as streets, rivers, contour lines or land use information. An example of such data can be seen in Fig. 4.1. Displaying and visualizing these data sets interactively in real-time 3D applications is a challenging task. In the remaining chapter we focus on the problem of rendering large-scale vector maps with many millions of line segments within a 3D real-time geo-visualization application. The massive amount of vector map data is challenging because of limited memory capacities on the *GPU* and because of the rendering time per frame which should be minimized for real-time purposes.

Previous vector map rendering methods may cause visual artifacts that negatively affect the interactive data exploration quality and corresponding geo-spatial analysis tasks. Examples of such artifacts are shown in Fig. 4.8. In particular, when combining multiple vector maps and a continuous multiresolution level of detail terrain mesh the mismatching resolutions cause significant artifacts in visualization. The problems in Fig. 4.8(a) occur because line segments of differing resolution vector maps (in yellow) float above or intersect the terrain. This unpredictable scene configuration makes the problem more complicated for geometric line rendering methods. Texture based approaches can solve the intersection problem, but suffer from other artifacts such as aliasing and projective distortions as shown in Fig. 4.8(b).

Compared to other vector map rendering methods, our approach performs a deferred shading pass for the vector map data on top of the traditionally rendered terrain surface, to avoid dependence on modified (preprocessed) vector map geometry. Consequently this bypasses the coupling between different vector maps or between vector maps and the terrain height-field during a preprocessing stage, and all data set combinations are changeable at runtime. Furthermore, we can avoid artifacts as shown in Fig. 4.8 but additionally, we achieve pixel-precise line display results even with multiple layers of vector maps with different level of detail resolutions being visualized on top of the terrain. Moreover, modern map visualization systems should allow the user to interactively change their map visualization with advanced vector styling methods. This requires a flexible line rendering method capable of extending the geometric line rendering to a line styling display concept. In this work we take these requirements into account and allow interactive modification of vector maps on top of the terrain visualization.

To achieve fast rendering, we will show how a deferred rendering approach can be exploited to interactively visualize large-scale vector maps with many millions

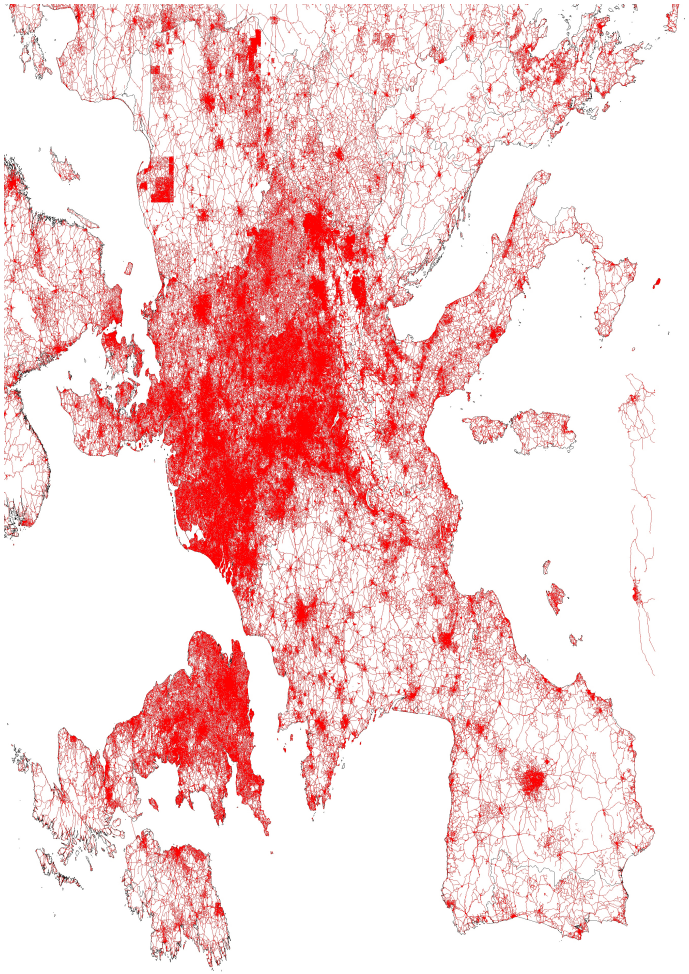
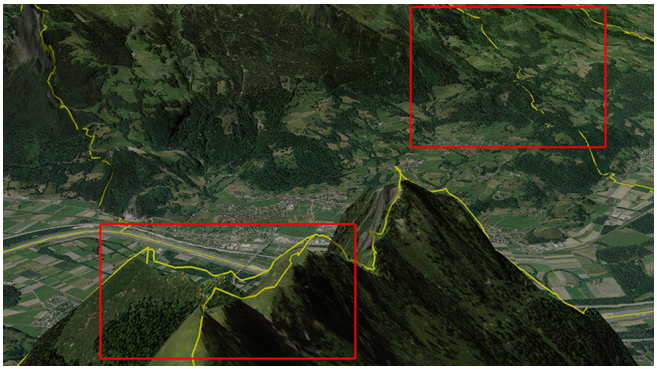
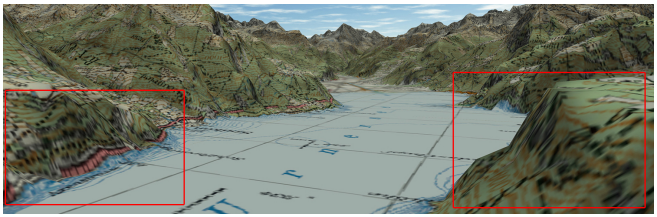


Figure 4.7: *Large-Scale Line rendering of the road map of Europe using traditional line geometry rendering.*



(a)



(b)

Figure 4.8: *Example artifacts when rendering vector maps. (a) Vector lines floating or intersecting the underlying 3D terrain, and (b) texture mapping based projection and resolution artifacts.*

of line features. In particular, we demonstrate how a clustered spatial line segment hierarchy can improve our deferred line rendering pipeline by optimizing the *GPU* workload for every pixel.

4.2.1 Related Work

In the following section we review state-of-the-art techniques for vector map rendering, as well as relevant work related to clustered deferred shading.

Vector maps as shown in Figs. 4.1 and 4.8 are usually line or polygon based data describing geometric objects with specific attributes. This geometric infor-

Criteria	Geometric Approach	Texture Mapping Approach	Shadow Volumes	Deferred Vector Maps
Rendering artifacts (Fig. 4.8)	Intersections z-Buffer artifacts	texture aliasing, distortions	geometric aliasing	geometric aliasing
Output accuracy	tesselation resolution	texture resolution	pixel-accurate	pixel-accurate
Dynamic changes	recompute tesselation	redraw textures	yes*	yes
Interactive styling & editing	recompute tesselation	evaluate texture content	missing geometry at shading	update line buffer
Memory consumption	geometry only	hi-res textures	geometry only	geometry only
Additional geometry needed	recompute tesselation	none	geometry extrusion	only line information
Preprocessing requirements	complete terrain necessary	texture hierachy	no preprocessing	line buffer generation
GPU requirements	none	none	geometry shader*	random memory reads
Applicable to large data	intensive preprocessing	expensive redraw preprocessed	pixel overdraw too expensive	demonstrated with $> 10^6$ lines

Table 4.1: *The table summarizes the differences between previous methods and our new technique (last column). *GPU requirements may be traded for a preprocessing step, making dynamic changes more costly.*

mation can be used directly for a 3D visualization as shown in [Bruneton and Neyret, 2008]. However, the most popular method is the combination of vector maps with image based information, like aerial photographic data, projected onto a terrain height field surface model. The closest related approaches for vector map visualizations discussed below can be divided into three categories:

- Texture Based Methods
- Geometric Subdivision Methods
- Shadow Volumes and Stencil Techniques

These methods are described in more detail in the survey of interactive visualization of vector data [Kersting and Döllner, 2002] and the survey of a digital earth [Mahdavi-Amiri et al., 2015]. A possible system description for these methods can be found in [Cozzi and Ring, 2011]. In Tab. 4.1 we summarize the main advantages and limitations of these three approaches in comparison to

our new deferred vector map visualization method. A common method used in geo-visualization systems is the texture based approach of rendering vector maps. Different descriptions of this method as well as comparisons to other methods can be found in [Kersting and Döllner, 2002; Wartell et al., 2003; Sun et al., 2008; Wang et al., 2009]. The basic idea is that vector maps are (orthogonally) rasterized to images and used as textures mapped on the terrain, e.g. as in Fig. 4.8(b). In general, any rendering system can easily apply textures to (terrain) surfaces, therefore, it is convenient and simple to implement such a texture based vector map visualization. In addition, texture based methods are often used if the development targets have limited hardware capabilities such as embedded devices, mobile platforms or browser based applications.

Texture Based Vector Map Rendering

Texture based vector maps, however, may suffer from artifacts as shown in Fig. 4.8(b). The highlighted artifacts are caused by the 2D texture projection as well as stem from the limited texture resolution. If the texture resolution is increased, however, more memory is needed. Many systems thus work with preprocessed texture pyramids, but adding higher resolutions increases the memory consumption and also the amount of texture files in these systems exponentially. Additionally, the texture pyramid may introduce border artifacts of the vector map information during rendering when used in a multiresolution level of detail system. Furthermore, most of the systems using this type of visualization do not allow immediate modification and styling of vector maps within an interactive 3D display session. To achieve a precise and suitable visualization for interactive vector map modification, it is necessary to manage a dynamic texture pyramid and to implement an on-the-fly rasterization step for updating vector maps. The amount of re-rasterization grows with the complexity of modification possibilities such as selection of vector map layers or highlighting of selected elements. Artifacts often appear when moving the view frustum close to the surface and the camera points to a far distance. In these cases, the texture based approach can get overly complex and costly in terms of memory, rendering time and system flexibility.

Geometric Subdivision for Vector Map Rendering

In geometric subdivision approaches for vector maps [Kersting and Döllner, 2002; Schneider et al., 2005; Xu et al., 2010; Deng et al., 2013] lines are subdivided according to the terrain mesh structure as illustrated in Fig. 4.9. Along the line segment at every change in slope of the underlying terrain, corresponding to crossing triangle edges, the line segment is subdivided. An application of the geometric line subdivision in combination with advanced map styling features can be found

in [Vaaraniemi et al., 2011] and [Wilkie et al., 2012]. The methods show possible ways to do precise line renderings.

However, this geometric approach for vector maps has the drawback that its line subdivision requires a predetermined and fixed combination of terrain triangulation and vector map subdivision. Dynamically changing terrain information, as is the case in continuous *LOD* terrain rendering, as well as unforeseeable combinations or editing of vector map data sets are hard to manage in this approach, and therefore, interaction possibilities are limited. This problem becomes even harder when the terrain information contains multiple layers. This is the case when height maps of different resolutions are combined and transitions between them are generated dynamically. It cannot be assumed anymore that a single point has a unique fixed height value, and often terrain blending is done in the shader stage so that the mesh cannot be retrieved. In such cases graphical artifacts would appear. Another aspect is the precise overlay of planar geometric objects, which often leads to z-buffer artifacts (*z-fighting*) due to the limited precision of the depth buffer.

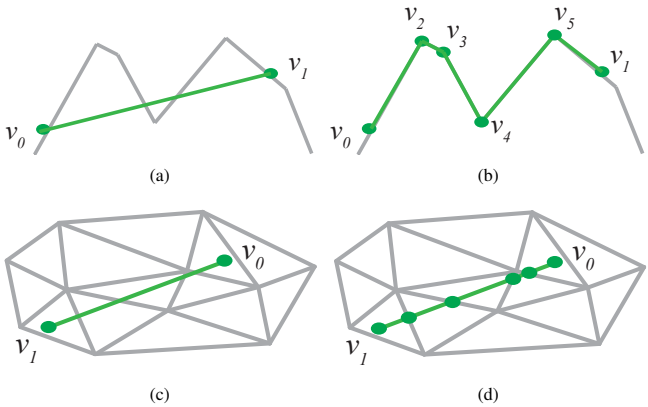


Figure 4.9: *Example for subdivision of a line according to the terrain height field. (a) Shows the elevation profile and (b) the subdivision. (c) Shows the line on top of the terrain and (d) the subdivision of the line matching the terrain mesh.*

Vector Map Rendering With Shadow Volumes

The shadow volume approach for vector map visualizations produces high quality solutions because the algorithm works independent from the terrain resolution and always produces a pixel-precise result on the screen [Dai et al., 2008; Wang et al., 2009; Yang et al., 2011]. The idea is that the geometry of a vector map is orthographically projected on the terrain by extruding the vector map's line segments into 3D polyhedral objects. The vertically extruded polyhedrons are then rendered in two steps, first the front faces and then the back faces. Analog to shadow volumes, every screen pixel counts the difference between front and back faces. The result per pixel then contains the information if this pixel is a part of a projected vector map or not.

The main drawback of this method is that multiple geometry rendering passes are required for the vector map, in addition to the terrain, and that the vector map geometry has to be extruded, e.g. in a geometry shader. The amount of geometry is thus about four times bigger than in the original vector map, and additionally, every extruded line segment has to be rendered twice. Furthermore, in case of large vector maps the vertically extruded geometry covers large portions of the screen and may produce a massive overdraw. Overall, this severely limits the technique to rendering very moderately sized vector maps of maybe a few thousand line segments. A special case is shown in [Ohlarik and Cozzi, 2011] where the method is optimized for vector maps only containing lines. Furthermore, it is hard to preserve the original vector map information so that advanced vector styling or procedural texturing can be achieved.

4.2.2 Deferred Shading

Our clustered deferred line rendering approach is inspired by the way lighting calculations are done in deferred shading pipelines [Saito and Takahashi, 1990; Liktor and Dachsbacher, 2012]. Deferred shading is applied to reduce the amount of shading operations by introducing a two-pass rendering pipeline. The first pass renders the geometry and produces a set of textures containing geometric scene information such as color, normal, depth or light information. The collection of output textures of the first pass is called a G-buffer, see also Fig. 4.10. All shading operations are done as image based effects using the information from the G-buffer in as few shading passes as possible. These render passes implement the effective shading and lighting for every pixel as well as other screen-space post processing operations such as antialiasing [Chajdas et al., 2011] or ambient occlusion [Bavoil and Sainz, 2008].

Building up a deferred shading pipeline raises the *GPU* memory consumption for additional texture layers and the pixel fill rate, because many more images

are produced than effectively used as final output frames. Eventually the overall rendering effort per frame can nevertheless be reduced drastically and allows for more image-space effects within one single frame. Clustered Deferred Shading described in [Olsson et al., 2012] subdivides the view frustum into clusters to improve the rendering speed for scenes with many lights. In our concept we took over the idea of clustering scene objects by creating line clusters in world-space as a preprocess.

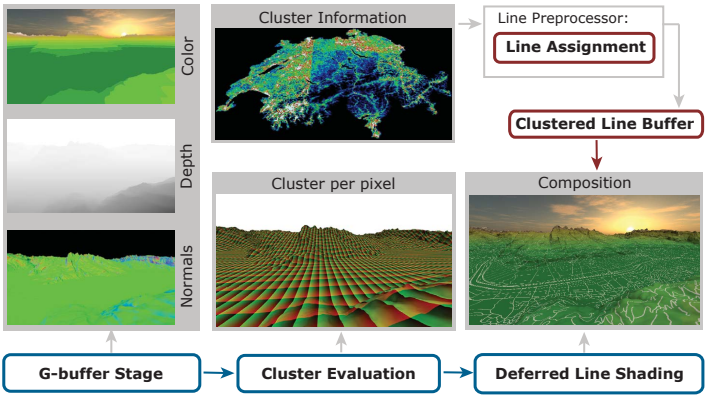


Figure 4.10: *Our deferred line rendering pipeline for large-scale vector map visualization. After the generation of the G-buffer, the cluster evaluation is performed and used as input in combination with the clustered line buffer for the deferred vector map line shading. The clustered line buffer is prepared in the line assignment preprocess.*

4.3 Deferred Vector Map Rendering

All approaches discussed above are using some kind of extracted geometry or geometry rasterized on textures for vector map visualizations. However, modern programmable *GPUs* allow the development of much more flexible systems. The basic idea of our line projection and rendering approach is to directly project and display vector maps on top of the terrain surface using an adaptation of the deferred shading principle without the need to generate intermediate geometric objects or textures. In contrast to common rendering methods where the color of a pixel is derived from the main (geometry) rendering pass, our approach inverts this principle for vector map visualization.

In Fig. 4.10 we outline the main steps of our deferred line rendering method as further detailed below. In a deferred shading stage, for every pixel corresponding to a point on the terrain it is determined if it contributes to the visualization of a vector map feature. Using the G-buffer data obtained from the main terrain rendering pass, we back-project each pixel into the 3D world and determine its location within the vector map, see also Fig. 4.13. To identify candidate line features, we use a clustered buffer storing all vector map line elements, which we call a *clustered line buffer*. An optimized search within the clustered line buffer clusters allows us to find the closest line feature quickly and color pixels accordingly.

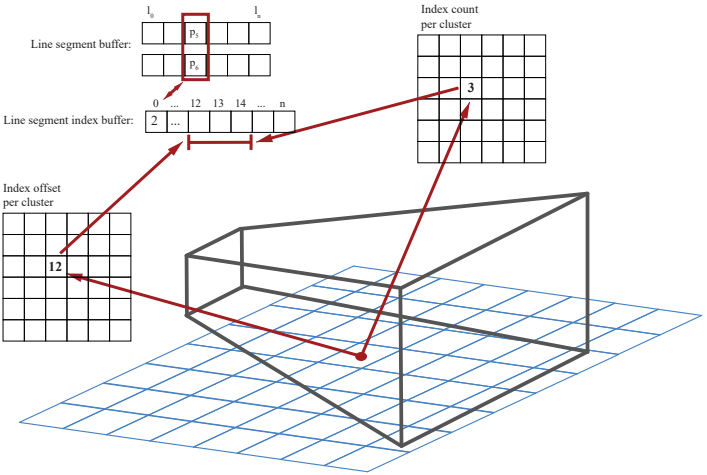


Figure 4.11: *The* clustered line buffer *GPU data structures consist of multiple array buffers and textures supporting efficient point-to-cluster location and point-to-line search queries from a given geographical location.*

4.3.1 Clustered Line Buffer

During the deferred shading stage, for each pixel the closest intersecting line feature, if any, must be determined very quickly. For this we use our *clustered line buffer* which *clusters* the individual line segments of the vector map features into a large 2D structure of cells. Every line segment is assigned to each buffer cell it

intersects. This can be a regular grid as currently implemented, but it could also be a multi-level nested grid or other space-partitioning hierarchy to better adapt to variations in the feature density in very large vector maps. Important is the ability to perform point-to-cell look-ups very efficiently, to allow fast identification of the cluster containing the closest lines potentially intersecting a back-projected pixel. Moreover, within each cluster the line segments are organized in an effective spatial search index structure to accelerate line search and pruning as well as minimize distance calculations after the coarse cluster identification.

The clustered line buffer is thus a data structure enabling fast point-to-line search queries and is designed to be used efficiently on the *GPU* during the deferred shading pass, as illustrated in Fig. 4.11. The clusters of this line buffer are formed during a preprocessing pass which assigns all vector map line segments to their corresponding clusters as further described below.

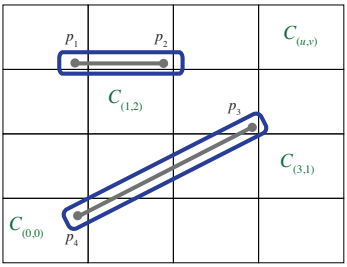


Figure 4.12: *Assignment of feature lines to clusters by rasterization.*

Two *line segment buffer* arrays store the start- and end-point coordinates $\{p_s, p_e\}$ of the individual line segments l_i on the *GPU*. A single *line segment index buffer* stores the indices to line segments concatenated for all cluster. The cluster grid is represented by two 2D integer textures, $o_{u,v}$ representing the cluster's *index offset* into the line segment index buffer and $c_{u,v}$ for the *index count* denoting the number of lines in the cluster. In a line assignment preprocess, for each cluster $C_{u,v}$ the vector map line features intersecting it are recorded, counted, and then concatenated to form the line segment index buffer.

The texture sizes for $o_{u,v}$ and $c_{u,v}$ equal the size of the cluster grid $C_{u,v}$. On the one hand the grid should not be too coarse, because that would include too many line segments within each cluster. On the other hand the amount of clusters is limited to the texture memory that can be committed. In our implementation we typically divided the map space into 256×256 clusters to express the cluster indices u, v as one byte each. Other multi-level nested grids or space partitioning

structures could be used as well, given a memory efficient implementation and fast cluster identification on the *GPU*.

The line assignment to a specific cluster follows a Bresenham line rasterization pattern as illustrated in Fig. 4.12, with certain line segments being assigned to several clusters. As the line drawing style is not known beforehand, we currently assume a predetermined conservative maximal line width which is incorporated into the line assignment. As indicated in Fig. 4.12, the line segment $l_1 = \{p_1, p_2\}$ lies in at least two clusters, but since the line has a certain line width we also have to assign its line segment index to all clusters it overlaps, e.g. $C_{1,2}$ too. Similar for $l_2 = \{p_3, p_4\}$ all overlapping clusters along the line are included, e.g. including also $C_{3,1}$.

To optimize the point-to-line query after the coarse point-to-cluster location, we use a hierarchical spatial index structure to organize all line segments within one cluster $C_{u,v}$. For simplicity combined with efficiency we generate a fully balanced binary bounding volume hierarchy (*BVH*). In the preprocess we first order all lines within each cluster based on their midpoint along a space filling curve. Then we build a binary tree bottom-up forming a balanced *BVH*. Other optimized *BVH* construction approaches could further improve upon this solution.

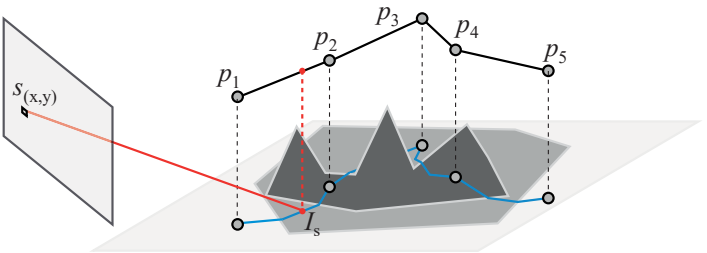


Figure 4.13: *Pixel back-projection and vector map location.*

4.3.2 Deferred Line Shading

The final fragment color for a screen pixel is determined in the *deferred line shading* stage, see also Fig. 4.10. In this stage our approach decides whether a pixel contributes to the visualization of a vector map element or not. Using the clustered line buffer, where all line segments are grouped into clusters, the search space of all line segments influencing a single pixel can be restricted to the line segments belonging to a certain cluster into which the pixel projects. Hence the line identification consists of two main steps: in the first step it is necessary to determine

the cluster in the clustered line buffer affecting a certain pixel. The second step determines if and which line effectively intersects the given pixel considering the applied line-style width.

The cluster of a certain pixel $s_{(x,y)}$ is determined by a backwards projection of the pixel position from screen space to world space coordinates. For every screen position $s_{(x,y)}$ a back projection can be applied using the corresponding depth $d_{(x,y)}$ from the G-buffer information, illustrated in Fig. 4.13 by the red line. The back projection of $s_{(x,y,d_{(x,y)})}$ then results in the point I_s which is the pixel's location in world coordinates. The back projection can be expressed as multiplication with the inverse view-projection matrix M_{VP}:

$$I_s = M_{\mathrm{VP}}^{-1} \cdot s_{(x,y,d_{(x,y)})} \tag{4.1}$$

With the information about the clustered line buffer's subdivision of the vector map and the point I_s in world space, it is easy to calculate the cluster $C_{u,v}$ containing the point I_s. In other words, this step maps the screen-space pixel locations (x,y) to the cluster-index space (u,v). In Fig. 4.10 we visualize this cluster evaluation by coloring each pixel in (red,green) based on its relative position within the corresponding cluster.

Using the back-projected $3D$ point location I_s of a pixel $s_{(x,y)}$ with depth $d_{(x,y)}$ from the G-buffer we thus have determined the cluster $C_{u,v}$ containing any potential line candidates. We now have to determine if the point I_s lies within a certain distance of any line segment $l_i \in C_{u,v}$ in the $2D$ vector map plane. As illustrated in Fig. 4.13, point I_s lies inside a certain distance to the line segment p_1, p_2 of the vector map. Thus it is considered to be part of that line segment and the pixel $s_{(x,y)}$ can be colored accordingly using the current style and visualization parameters.

For large and complex vector maps as shown in Figs. 4.16(a), 4.17(a) and 4.18(b), however, per-pixel line identification and point-to-line distance tests can become costly in our deferred line shading approach and some further optimizations are called for as described below.

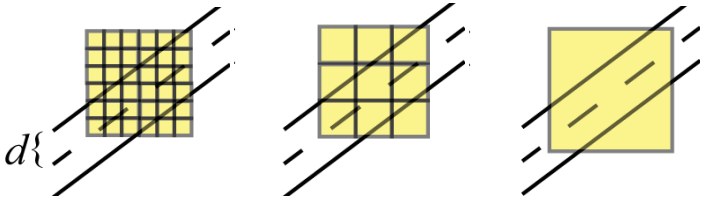
The point-to-line search and distance calculation within a cluster $C_{u,v}$ should be optimized to keep the per-pixel computation cost low. Fig. 4.17(b) shows a heatmap indicating the varying cost effort to find corresponding line segments. Given the varying number of line segments in different clusters, for large vector maps we organize the line segments within each cluster in a BVH as already mentioned in Sec. 4.3.1 to limit the number of distance test computations.

Given a pixel's position I_s in world coordinates and in vector-map space, the BVH can be traversed effectively to identify the leaf nodes containing any line segments $l_i' \subseteq C_{u,v}$ which are potentially closer than a certain given distance from I_s. Only for this subset of lines l_i' the point-to-line distance has eventually to be computed. Therefore, even for very large vector maps with many millions of

lines, the amount of line distance tests required per cluster is eventually reduced to a small number and can thus be performed in a fragment shader for each pixel efficiently.

4.3.3 Line Styles and Antialiasing

The deferred line rendering method outlined above can further support visualization features beyond bare line rendering. In particular, the screen-space per-pixel shading allows the implementation of advanced colorization decisions like applying different line styles and line patterns on-the-fly during interactive rendering. Cross-sectional color patterns can easily be incorporated into a generic line shader taking the width of the line feature and the distance of the pixel to the line into account (see also Fig. 4.19 and results in the next section). Longitudinal procedural patterns can be applied using pattern buffers and more complex line shaders as well. Furthermore, given the pixel-to-line distance to the closest or all pixel-intersecting line features, alpha-blended compositing of multiple features or over background can be achieved. Dynamically varying or view-dependent visualization parameters can also be incorporated into the deferred line shading process, as demonstrated e.g. in Fig. 4.15 with an interactive lens function.



(a) d > pixel (b) d ≈ pixel (c) d < pixel

Figure 4.14: *Three cases where aliasing artifacts can occur: (a) Regular line rendering staircase artifacts appear at distances where the line width d is bigger than the pixel size. (b) If d is almost equal to the pixel size artifacts may appear by wrong hits in the pattern buffers when using line-style patterns. (c) If d is smaller than the pixel size parts of a line segment are hidden and wrongly identified as background.*

Thin lines can suffer from aliasing artifacts as shown in Fig. 4.17(a) where distant contour lines at the back become discontinuous. This effect may appear in several ways and is caused by the difference of the line size and the size of one pixel at this distance. In our manual *GPU* implementation, we distinguish three cases as described in Fig. 4.14. In the first, normal case when the line width covers several pixels, normal pixel drawing works and common artifacts can be

reduced by supersampling or other standard antialiasing algorithms. The second case describes artifacts due to procedural patterns, see Fig. 4.14(b), which are not easily solved even when using alpha-blending techniques. We handle this case by reverting to the main color of the line pattern for line segments at far distances to convey the primary association to feature categories. The result can be seen in Fig. 4.16(b). The third case corresponds to a line segment missing the line test for some pixels due to the line width being smaller than the pixel at far distances. This is similar to rendering phone wires as described in [Persson, 2012]. The main idea is to adjust the line width for distant points and color the pixel using alpha-blending based on the line's approximative coverage of the pixel. This is estimated from the ratio between the line's width and the pixel's extent.

4.3.4 Results

Data set	Vector map Lines	Cluster max. size	Line Assignment	Render time
Vorarlberg (higher streets)	193,042	1,106	9.8 s	50 ms
Switzerland (TLM streets)	16,556,412	3,429	101.7 s	55 ms
Carinthia (isolines)	31,297,095	4,650	120.7 s	190 ms

Table 4.2: *Dataset overview. For each vector map we used a fixed cluster grid of* 256×256*. Average render times measured for viewpoints (3840* $\times$ *2160 for 2* $\times$ *supersampling) shown in Fig. 4.16(b), 4.16(a), 4.17(a) including* $\sim$*15 ms for rendering terrain.*

Our implementation runs on an Intel Core i7 3.5 GHz, 16 GB RAM, Nvidia GTX680 (4GB RAM, 1920×1080) machine with C++ and OpenGL. It allows us to load and interactively visualize large-scale terrain and vector map datasets. In particular, the system supports exploration of large-scale vector maps interactively in a full 3D environment. Tab. 4.2 lists the vector map datasets used in our tests. Experimental results show that our approach can be used for large-scale interactive vector map visualization, see also Figs. 4.16, 4.17 and 4.18.

In comparison to texture-based visualizations, it can be guaranteed that the visual result is a pixel-precise rendering as demonstrated in Fig. 4.19. Our system maintains full pixel precision at any zoom-in factor even near to the terrain. To achieve this with texture mapping it would require a much more complex and elaborate solution still suffering from resolution artifacts, see also Fig. 4.19(f). In contrast to geometric line rendering approaches, intersecting or floating line artifacts as shown in Fig. 4.8(a), as well as z-fighting problems as shown in Fig. 4.19(a) can be avoided.

Our deferred line rendering solution also efficiently supports interactively changing visualization parameters such as line size or styling properties as well as view-dependent data selection without reloading or re-rasterization of textures. A dynamic view-dependent vector map data-lens example is shown in Fig. 4.15.

A regular grid based clustered line buffer without additional hierarchical line segment organization is capable of handling vector maps of up to 200,000 line segments and 1,100 lines per cluster. The use of hierarchical spatial line indexing within each cluster further makes the interactive exploration of much larger vector maps possible. Vector maps with several millions of line segments can be visualized at interactive frame rates with our methods using the maximal cluster sizes indicated in Tab. 4.2. The rendering effort per pixel depends on the number of lines and their organization within each cluster in the line buffer. Fig. 4.10 shows an example for the content of the cluster information. Fig. 4.17(b) illustrates the per-pixel line search and distance test costs based on our clustered line buffer and local bounding volume line hierarchies.

4.4 Conclusion

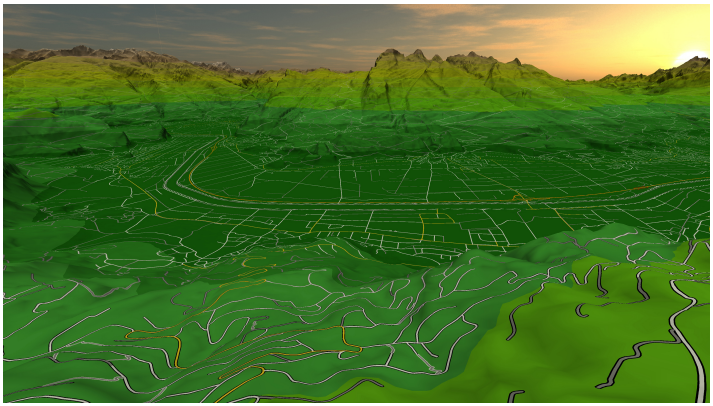
In this chapter we present a novel approach for handling GIS vector data sets within interactive 3D environments. Our approach represents an efficient and flexible solution for different purposes. It can be used for interactive editing and adaptive or view-dependent styling of vector maps as well as for large-scale visualizations. In particular, it is also suitable for dynamic level of detail and out-of-core geographic visualization systems. Our approach relies on multiple rendering passes and was implemented using modern graphics hardware. However, as hardware is continuing to improve in terms of graphics functionality, this limitations may be overcome in the near future.

Our next steps are the extension of the system to work with polygonal objects as well as further improve rendering speed. In addition, future plans include an extensive evaluation and incorporating the system with a vector map out-of-core level of detail system to make larger data sets accessible.

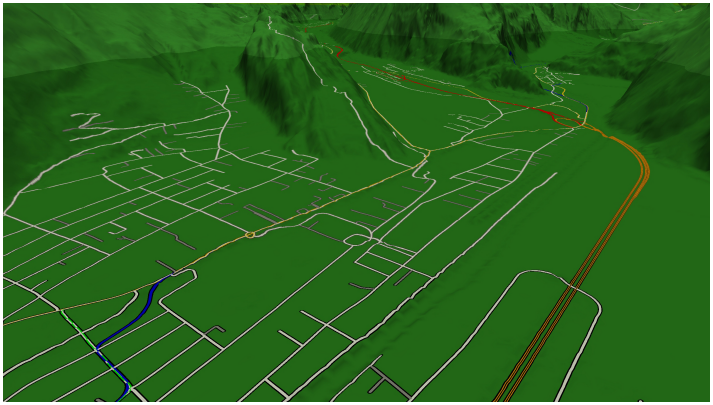
(a)

(b)

Figure 4.15: *Example for interactive editing functionality. It is possbile to interactivly fade out specific categories of streets from the vector map on a pixel-precise basis using alpha blending.*

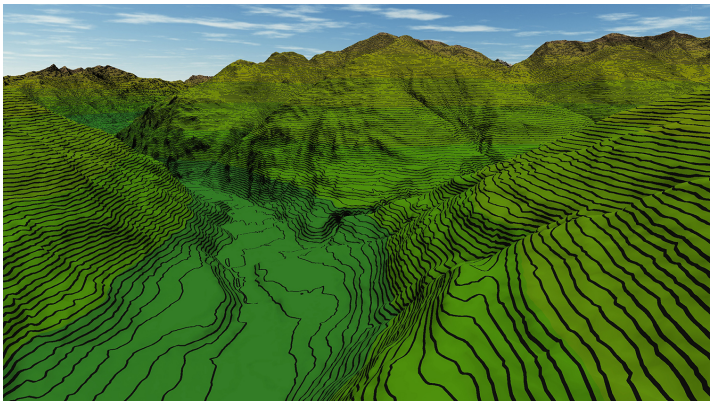


(a)

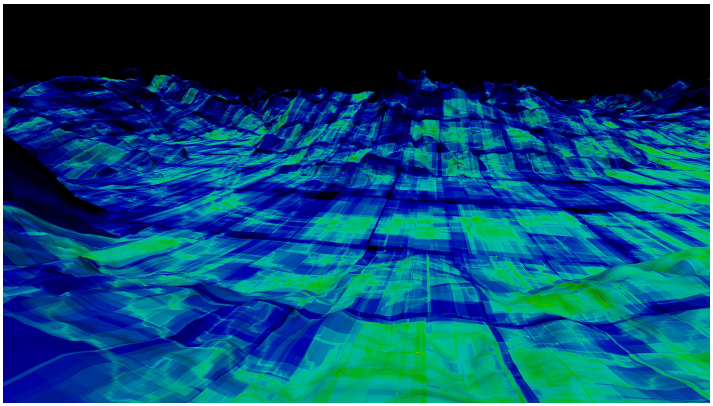


(b)

Figure 4.16: *(a) An example for line rendering of street data with styling pattern. (b) Example showing the smooth transition of a procedural pattern to solve alising artifacts shown in Fig. 4.14(b).*

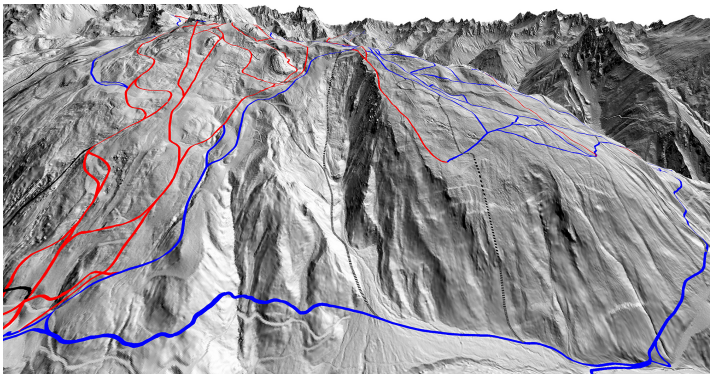


(a)

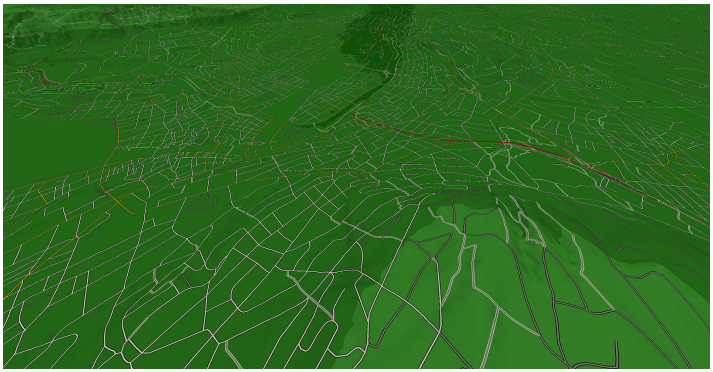


(b)

Figure 4.17: *(a) Carinthia isolines with black contour lines and distant lines suffering from aliasing artifacts. (b) A heatmap illustrating the bintree line search and intersection traversals cost from black (zero traversals) over blue to green.*

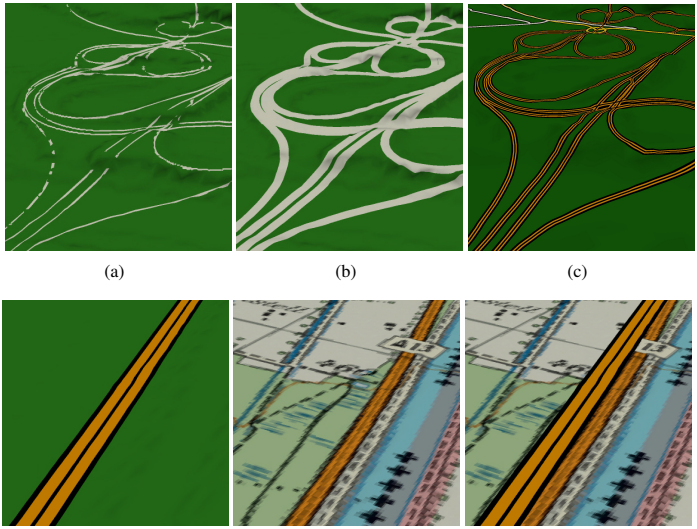


(a)



(b)

Figure 4.18: *(a) Visualization of ski slopes over a hill-shade textured terrain with blue and red colored slopes for beginner and advanced levels respectively. Off-piste tracks are shown in a stippled black style pattern. (b) Swiss TLM street data containing 21 different categories of streets and using multiple line styles. Streets smaller than 4m wide using white and grey combinations, streets with 6m width are shown in yellow, and streets more than 10m wide are shown in red. Highways are highlighted in orange.*



(a) (b) (c)

(d) (e) (f)

Figure 4.19: *Comparison of our approach (b,c,d) with normal 3D geometry (a) and texture based rendering (e,f): (a) A street rendered as a simple geometric object. (b) Simple line rendering with our method, and (c) applying some advanced vector styling. (d) A street rendered with our algorithm, and (e) using texture mapping at the highest resolution available for this texture set. (f) Overlay of (d) and (e) for direct comparison.*

CHAPTER 5

PATH BUNDLING

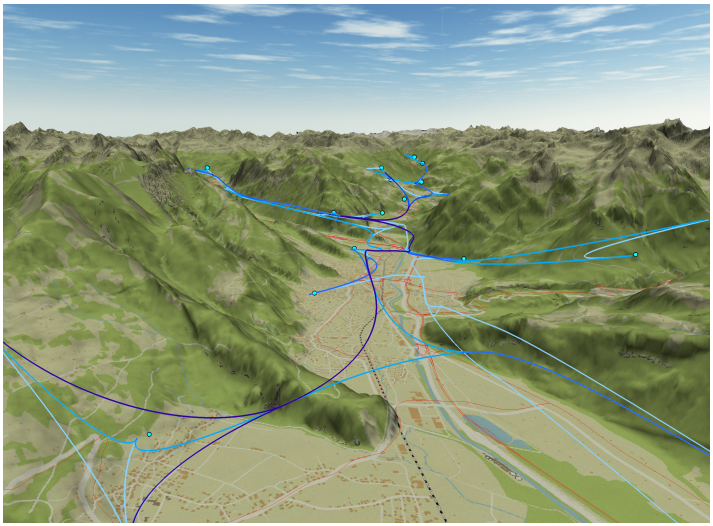


Figure 5.1: *Perspective view showing bundled paths over 3D terrain of Vorarlberg.*

5.1 Introduction

An interesting challenge in geographic visualization is the analysis of transportation relations. These relations are represented by dense graphs containing many point-to-point connections. The edges represent semantic information, such a commuter connection between towns or an airline connection between two airports. The geographical location for this traffic relations allows the visual identification of traffic highlights, such as traffic jams or very frequently visited nodes. These traffic relation graphs can become large graphs with several thousands of connections and suffer from cluttering effects. An example of such as graph is shown in Fig. 5.2)

Effective visualizations of large-scale and dense graphs or networks commonly use so called *graph bundling* techniques. The goals of graph bundling are first the reduction of visual clutter in a dense graph dataset, and second the highlighting of major connectivity paths and patterns (see e.g. Fig. 5.11 and 5.12) within this dataset. Individual as well as groups of related connections in a dense graph are very hard to identify. Bundling of paths leads to a visualization which makes it easier to analyze the prevalent connectivity structures.

Bundling dense point-to-point graphs, such as commuter data, may have to be constrained to and visualized over a variety of complex vector map reference networks, such as roads or train routes. This additional vector map often is the bases for some *routing* constraints that map the dense point-to-point connections to paths over the reference network. A visualization of such paths following the reference network constraints enriches the effective information visualization and visual analysis task. However, the density of the paths and complexity of the underlying reference network may cause visual cluttering and loss of the larger context of connectivity information.

Reducing clutter and highlighting major structural link information is even more important and challenging in an interactive geo-visualization when using a 3D environment. In this work we assume paths and graphs having nodes and edges with 3D coordinates within a geospatial framework including 3D terrain information and 3D obstacles. In 3D, it becomes harder to analyze, identify and bundle path trajectories in a dense graph constrained to a reference network due to the added interference, i.e. intersections and occlusions from 3D objects and obstacles.

In this chapter we present a novel vector map constrained path bundling method, that takes a dense graph over spatial locations and generates a variable level of detail bundling constrained to a reference vector map while avoiding intersections with given obstacles in 3D. See also overview shown in Fig. 5.1. In particular, we demonstrate our approach based on dense geo-spatial point-to-point relations that are routed and bundled over geographical vector maps, all embedded and visual-

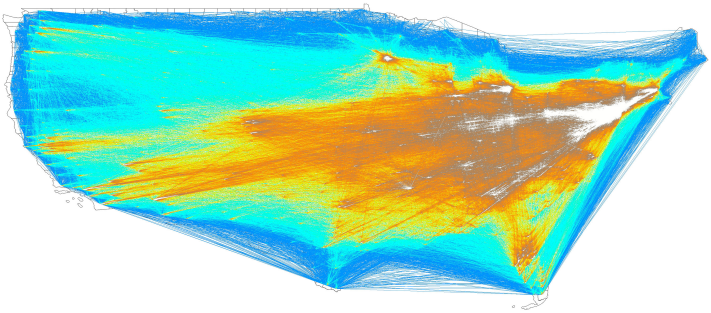


Figure 5.2: *Example for a large-scale graph: US Migration Graph from* 2012.

ized in a 3D digital terrain elevation model. Furthermore we are going to compare our results with state-of-the art techniques to the extent to which they may be applicable.

5.2 Related Work

A general overview of clutter reduction in graphs and information visualization is discussed e.g. in [Ellis and Dix, 2007] and taxonomies for graph visualizations are described in [Lee et al., 2006].

Clutter reduction techniques can be classified as edge or node based problems. Of these, the former applies to this work as the geo-referenced start- or end-points are intentionally left unmodified. Edge bundling methods can further be subdivided into two categories, geometry or image based methods. Geometry based methods use and manipulate the graph edges on the geometric object level, whereas image based methods solve the problems primarily in discretized image space and see the problem in a 2D planar projection. Our method belongs to the category of geometry based edge bundlings. However, in contrast to other methods, our aim is a 3D constrained geometric bundling suitable for interactive visualization in a 3D environment including correct visibility and depth occlusions in perspective 3D views. Past published geometric graph bundling methods include force-directed edge bundling (FDEB) [Holten and van Wijk, 2009], divided-edge bundling [Selassie et al., 2011], multilevel agglomerative edge bundling (MINGLE) [Gansner et al., 2011], winding roads [Lambert et al., 2010c], geometry-based edge clustering for graph visualization [Cui et al., 2008] and hierarchical

edge bundles [Holten, 2006].

FDEB [Holten and van Wijk, 2009] applies a field force to subdivided segments of a graph, so that subdivision points can be moved according to their attracting force directions. The algorithm has a high algorithmic complexity, but also avoids any complex hierarchical data structures. An extension of FDEB using different optimization constraints is described in [Selassie et al., 2011]. The idea of force-directed edge bundling can be extended to 3D as recently shown in [Zielasko et al., 2016]. However, the main challenge is the identification of edge compatibility measurements which are working for all 3D line situations when using a graph dataset with 3D nodes and edges.

The MINGLE algorithm [Gansner et al., 2011] is also based on the idea of optimization similar to FDEB. In MINGLE, the optimization is done with respect to limiting pixel overdraw, also called ink minimization, according to neighborhood information extracted from a 4D line interpretation. The method is proven to be faster than FDEB, however, the ink saving algorithm assumes that all edges merge at predefined control points, which can produce esthetically insufficient results.

Another approach for edge bundling is the usage of control hierarchies. Geometry-based graph bundling [Cui et al., 2008] uses a control mesh forcing edges to pass through control points. The edge data is used to extract a control mesh and find a 2D partitioning layout, but the artificial origin of this control mesh does not imply any semantic meaning to the final result. Geometry-based graph bundling could possibly be extended to 3D datasets using a more complex control mesh and smoothing algorithm, but it is not clear how. Hierarchical edge bundles described in [Holten, 2006] are using hierarchical tree or graph structures. This approach is well known to visualize nodes in a hierarchical correlation. The application of hierarchical edge bundles to a 3D visualization can be found in [Caserta et al., 2011]. However, visualizing hierarchical graph or tree structures is a different problem setup from normal graphs or applying a constraint network and thus not directly applicable or comparable.

Similar to the previous approaches, the Winding Roads algorithm described in [Lambert et al., 2010c] uses an artificial grid for routing bundles. The algorithm shows examples with a Voronoi grid structure and a quadtree grid structure. The edge routing is done based on a shortest path search. In our work we adapt and extend the idea of this algorithm in a more general way. We will show how the idea of edge routing can be applied to general reference graphs and we will extend this approach with a 3D line level of detail (LOD) method. There is another extension to the Winding Road algorithm showing one possible way of a 3D impression [Lambert et al., 2010b]. This extension uses a 2D approach and the final results are done using bump mapping for the 3D impression. However, bump mapping is a 2D effect for a 3D impression and it will not be possible to visually separate bundles in 3D environments if needed for example for skew lines.

The bump mapping effect can be applied on almost any method, including image based methods. An example is shown in [Hurter et al., 2012]. Recent and important image based methods are: graph bundling by kernel density estimation (KDEEB) [Hurter et al., 2012] or skeleton based edge bundling (SBEB) [Ersoy et al., 2011].

KDEEB [Hurter et al., 2012; Hurter et al., 2013b; Hurter et al., 2013a] is based on the idea of using the spatial edge density for clustering of graph edges, streaming graphs or graph sequences. This can be done very fast and does not require any special data structures. The method also shows some results for obstacle avoidance. SBEB [Ersoy et al., 2011] use distance fields to iteratively attract edges against a skeleton interpretation of the graph. These distance fields are generated by a 2D clustering of the graph edges and are used to define attracting forces for each edge.

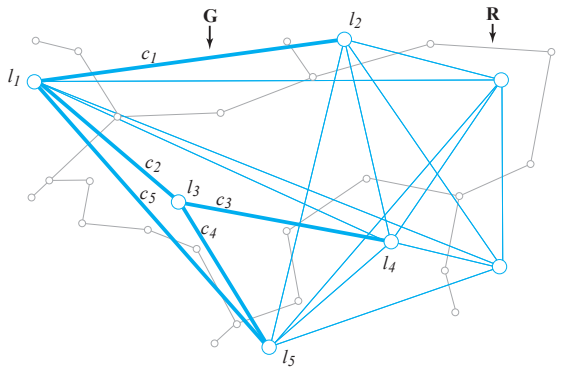
In general, image based methods can be very efficient. The main challenge when applying image based algorithms to 3D are the inherent rasterization steps. Image based models tend to project the 3D spatial information to a 2D case as well as cut away the geometric precision by rasterization. These steps can often be incorporated into a system as such. Eventually, visibility and depth-occlusion correct perspective 3D views may not easily be supported within such a visualization system.

In contrast to previous graph bundling approaches, our method works with geometric objects within a 3D environment, taking into account visibility and depth-occlusion problems, such that LOD-based interactive graph visualization can fully be integrated in 3D viewing applications. Moreover, recent publications have shown that interactive exploration of large graph datasets is crucial for effective online analysis. See also e.g. [Wong and Carpendale, 2005; Lee et al., 2006; Guo, 2009; Lambert et al., 2010a; Luo et al., 2012; Zinsmaier et al., 2012] for example work on the aspect of interactivity in graph visualization. Our work also focuses on interactive and adaptive LOD constrained path bundling and visualization.

5.3 Constrained Graph Bundling

5.3.1 Input Graph and Reference Vector Map

Our input data consists of a dense graph $\mathbf{G} = (L, C)$ with spatial locations L and many point-to-point connections $C \subseteq L \times L$. Typical representatives of such dense graphs are shown in Figs. 5.11(a), 5.13(a) and 5.14(a). Additionally we have a reference vector map, or reference graph $\mathbf{R} = (V, E)$ which defines a (planar) network of edges E defined between its vertices V. An example of such a reference network is shown in Figs. 5.15(a). Furthermore, we define a trail $P = (V', E')$



(a)

Figure 5.3: *Example input graph* **G** *(blue) over reference network* **R** *(grey) with a few point-to-point connections* $c_{1...5}$ *highlighted.*

in the reference graph **R** as a path of consecutive edges $E' = \{e'_0 \dots e'_{k-1}\} \subseteq E$ connecting vertices $V' = \{v'_0, \dots v'_k\} \subseteq V$ with $e'_i = (v'_i, v'_{i+1})$. See also Fig. 5.3 and 5.4 for an example of input graph, reference network and trail paths.

5.3.2 Bundling over Reference Graph

The main goal of our approach is the combination of the above defined graph datasets **G** and **R** by bundling all the connections in **G** constrained to trajectory paths P over the reference network **R**. For this we first map the input locations $l_j \in L$ to their corresponding nodes $v_{j_L} \in V$ in the reference graph, and we find for each connection $c_i \in C$ a corresponding path P_i in **R**. Thus over the reference graph **R** we use a set of trails $\mathbf{T} = \{P_i\}$ implied by C from the input graph **G**.

By representing each graph connection c_i by a path P_i in **R** we achieve a constrained bundling since the general cluttered connections C in **G** are now mapped onto a constrained set of path trails **T** over **R**. The actual bundling in fact occurs due to the fact that multiple connections $c_{i_1,\dots i_k}$ are eventually mapped onto trails $P_{i_1,\dots i_k}$ which share sections of their paths in the reference network, see also Fig. 5.4.

For adaptive visual bundling and interactive visualization of the constrained bundling **T** we display all its paths P_i as variable LOD B-splines. The effect of the

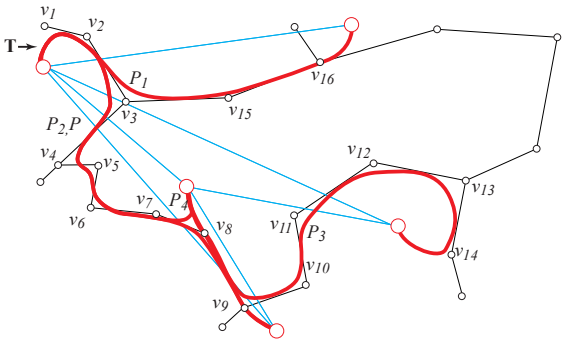


Figure 5.4: *Connections $c_{1...5}$ bundled by B-spline curves, controlled by the vertices of the corresponding trails $\mathbf{T} = \{P_{1...5}\}$ in the reference network* $\mathbf{R}$.

constraints and thus the visual outcome can be adjusted interactively at runtime, hence the bundling can be constrained more or less to the underlying reference network, and hence it could also be related to the accuracy of path P_i representing the connection c_i in the reference network $\mathbf{R}$.

Example vector map constrained bundlings are shown in Figs. 5.12(b), 5.13(b), 5.14(b) or 5.15(b) which use major roads, train rails or air corridors reference networks to bundle commuter, migration or flight data. In these examples, given the most likely point-to-point routings, the constrained bundling shows the potential impact of people movements or flights influencing the traffic on the different transportation network segments. With our method the transportation streams are constrained to and bundled along a reference network to follow realistic traffic paths routes, instead of bundling streams purely visually in 2D image or 3D object space.

Our algorithm consists of six preprocessing steps as indicated in Fig. 5.5. For performance reasons, the reference network $\mathbf{R}$ is first processed and simplified to a reduced graph $\widetilde{\mathbf{R}}$, minimizing the number of degree two nodes. A kd-tree index structure is then built that supports finding the nearest nodes in $\widetilde{\mathbf{R}}$ for all locations L of $\mathbf{G}$. Between these start- and endpoints, for each edge c_i the shortest path $\widetilde{P}_i$ is computed in the reduced network $\widetilde{\mathbf{R}}$. Note that any other point-to-point routing according to the reference network semantic and properties could be used for this step to suit a particular application. Thereafter, a quadratic B-spline curve is defined with control points from the corresponding vertices of P_i in the origi-

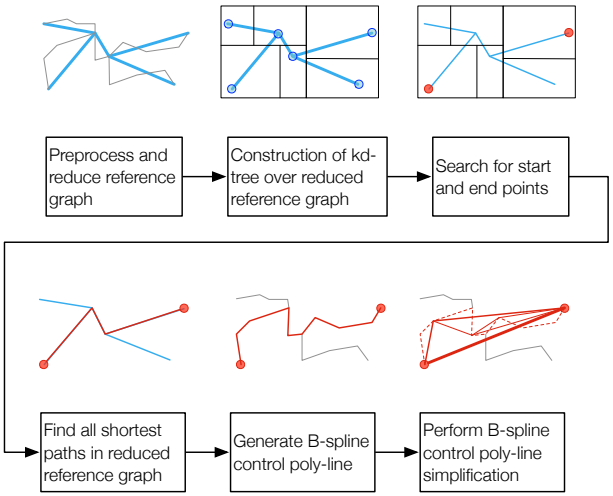


Figure 5.5: *Overview of the preprocessing steps for our reference vector map constrained graph bundling.*

nal network **R**. Based on additional obstacle avoidance constraints, hierarchical variable LOD B-splines are constructed using an iterative polyline simplification method.

Reference graph reduction

We first simplify the reference graph to cope with a large number of input locations L and point-to-point connections C, so that costly point location and (shortest) path queries on detailed and complex reference networks **R** are avoided. The reduced undirected weighted reference graph $\widetilde{\mathbf{R}}$ is obtained by merging degree-two nodes such that $\widetilde{\mathbf{R}}$ eventually only consists of endpoint and bifurcation nodes of **R** as illustrated in Fig. 5.6. Application specific edge weights and attributes, such as lengths, must be aggregated during these merging steps.

The reference graph reduction is possible since for multiple trails P_i sharing the same section $\{v_j \dots v_{j+k}\}$ of a path between two bifurcation or endpoints v_j and v_{j+k}, that section can be collapsed to a new reduced edge $\widetilde{e}_{jk} = (v_j, v_{j+k})$. Note that shortest path queries over these collapsed edges $\widetilde{e}_{jk}$ are not affected as

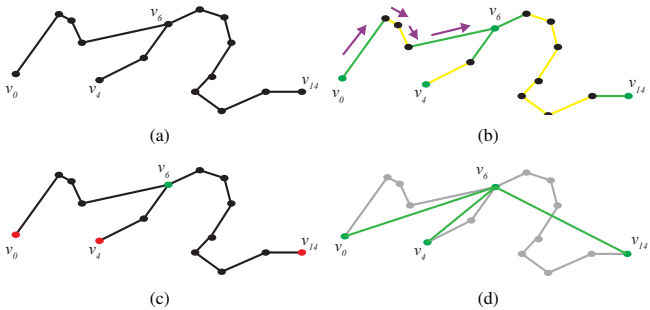


Figure 5.6: *(a) Complete reference network* $\mathbf{R}$*. (b) Traversal of reference network identifies endpoints and bifurcation points. (c) Endpoints (red) and bifurcation points (green) are retained, and degree-two points (black) can be removed. (d) Reduced reference graph* $\widetilde{\mathbf{R}}$ *shown in green.*

long as the path length of the sequence $\{v_j \dots v_{j+k}\}$ is retained as aggregated edge weight for $\widetilde{e}_{jk}$.

Vector maps are often stored as a group of polylines from which the reference graph network $\mathbf{R}$ must first be built. This involves identifying all polyline endpoints and where they connect to other polyline vertices. In Figs. 5.6(b) and 5.6(c) 3 polylines $\{v_0 \dots v_6\}$, $\{v_4 \dots v_6\}$ and $\{v_6 \dots v_{14}\}$ are merged and the endpoints and crossings are marked. The reduced graph $\widetilde{\mathbf{R}}$ is then formed by omitting the polylines' degree-two vertices in between the marked nodes as in Fig. 5.6(d).

Nearest neighbor and shortest path queries

The next steps consist of mapping all locations L of G to their nearest points in $\widetilde{\mathbf{R}}$ and finding the corresponding paths for all connections C. Instead of the shortest path finding described here, more specific application dependent routing rules could be applied for path finding in $\widetilde{\mathbf{R}}$. Using a kd-tree over the vertices of $\widetilde{\mathbf{R}}$ we can quickly find the vertices $\widetilde{v}_{i_{\text{start}}}, \widetilde{v}_{i_{\text{end}}} \in \widetilde{\mathbf{R}}$ corresponding to the endpoints $l_{i_{\text{start}}}, l_{i_{\text{end}}} \in L$ of each connection $c_i \in C$.

For each connection c_i we are able to find the shortest path $\widetilde{P}_i = \{\widetilde{\mathbf{p}}_{i_{\text{first}}}, \dots \widetilde{\mathbf{p}}_{i_{\text{last}}}\}$ in $\widetilde{\mathbf{R}}$ between $\widetilde{v}_{i_{\text{start}}}$ and $\widetilde{v}_{i_{\text{end}}}$ using Dijkstra's shortest path algorithm. Many optimized and hardware accelerated methods exist to improve this step (see e.g. [Goldberg and Harrelson, 2005; Delling et al., 2011]). Given all the paths $\widetilde{P}_i$ in the reduced graph $\widetilde{\mathbf{R}}$, the entire trails $\mathbf{T} = \{P_i\}$ in the complete reference network $\mathbf{R}$ can

then easily be found by path expansion $\widetilde{P}_i \rightarrow P_i = \{\mathbf{p}_{i_{\text{first}}}, \ldots \mathbf{p}_{i_{\text{last}}}\}$ to also include the left-out degree-two vertices.

These trails define polylines through $\mathbf{p}_{i_{\text{first}}}, \ldots \mathbf{p}_{i_{\text{last}}}$ which will serve as the control polygons for variable LOD B-spline curves as described below. Another advantage of our method, over other generic graph bundling approaches is, that until this point all information is in 3D, i.e. the control points $\mathbf{p}_i \in \mathbb{R}^3$, and the B-spline interpolation and display can thus be done fully in 3D.

B-spline trail interpolation

For visual bundling and display of trails $P_i = \{\mathbf{p}_{i_{\text{first}}}, \ldots \mathbf{p}_{i_{\text{last}}}\}$ we use *endpoint interpolating quadratic* B-spline curves $s_i(u)$. The control points $\mathbf{p}_k$ are given by the trail P_i (with its point indices $i_{\text{first}} \ldots i_{\text{last}}$ remapped to $k = 0 \ldots K$), and the quadratic B-spline of a trail path P_i can then be written as:

$$s_i(u) = \sum_{k=0}^{K} \mathbf{p}_k \cdot N_k^2(u) \tag{5.1}$$

With the B-spline basis functions $N_j^d(u)$ given below, a quadratic B-splines ($d = 2$) can interpolate the endpoints $\mathbf{p}_0, \mathbf{p}_K$ by setting the knot values $u_0 = u_1 = u_2 = 0$ and $u_{K+1} = u_{K+2} = u_{K+3} = 1$, with the remaining $u_{j=3,\ldots K}$ of the knot vector u_j being uniformly spaced.

$$\begin{aligned} N_j^d(u) &= \frac{u - u_j}{u_{j+d} - u_j} N_j^{d-1}(u) + \frac{u_{j+d+1} - u}{u_{j+d+1} - u_{j+1}} N_{j+1}^{d-1}(u) \\ N_j^0(u) &= \begin{cases} 1 & u \in [u_j, u_{j+1}] \\ 0 & otherwise \end{cases} \end{aligned}$$

Due to partition of unity property and the fact that $N_j^d(u) = 0 \quad \forall u \notin [u_j, u_{j+d+1}]$, degree-$d$ B-splines curve points $s_i(u)$ with $u \in [u_j, u_{j+1}]$ are strictly contained within the *convex hull* of the $d+1$ control points $\mathbf{p}_{j-d}, \ldots \mathbf{p}_j$. This property is important for our 3D intersection constrained variable bundling as outlined below, and for quadratic B-splines it means that a curve point for $u \in [u_j, u_{j+1}]$ is guaranteed to lie within the triangle $\triangle_{j-1}$ defined by the points $\mathbf{p}_{j-2}, \mathbf{p}_{j-1}$ and $\mathbf{p}_j$.

Constrained B-spline simplification

In a 3D environment, the bundled trail paths P_i displayed as B-spline curves as described above may interfere with other 3D objects and obstacles. In particular, stream bundles following transportation network lines may intersect with the

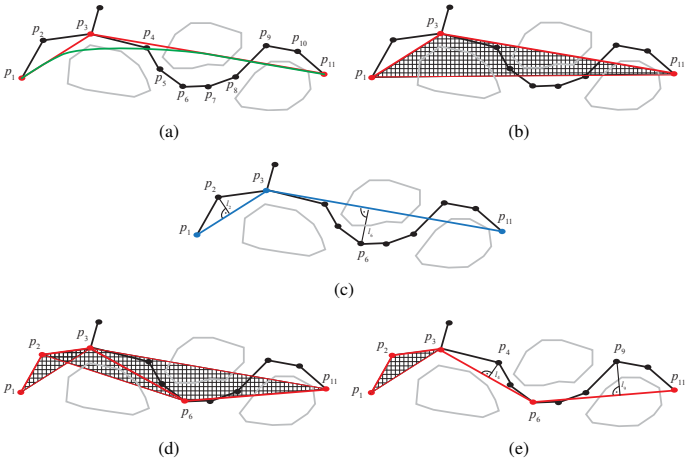


Figure 5.7: *Example for a constrained quadratic B-spline control polygon refinement around obstacles. (a) Original B-spline (green) with minimal control polygon (red). (b) Intersection test of minimal control point triangle. (c) Douglas-Peucker refinement of the affected two line segments. (d) Recursive test of refined triangles for obstacle intersection. (e) Termination for one and recursive refinement of two other line segments.*

terrain height-field in a 3D geographic visualization system (see e.g. Figs. 5.9 and 5.10). In the following we describe how 3D obstacles can be taken into account to generate non-interfering quadratic B-splines, and more specifically, how we can define variable LOD B-splines around terrain height-field constraints in 3D.

Occlusion queries in 3D scenes can effectively be implemented using graphics hardware [Wimmer and Bittner, 2005]. For our approach we use hardware occlusion queries to define constrained B-splines through iterative polyline refinements using the Douglas-Peucker algorithm [Shi and Cheung, 2006].

The principle of the Douglas-Peucker algorithm is to reduce a polyline through a point sequence $\{\mathbf{p}_0, \ldots \mathbf{p}_K\}$ initially to the line segment $\overline{\mathbf{p}_0, \mathbf{p}_K}$. Then the next point $\mathbf{p}_m$ with $0 < m < K$ is added which has the largest distance to the line $\overline{\mathbf{p}_0, \mathbf{p}_K}$. The resulting polyline $\{\mathbf{p}_0, \mathbf{p}_m, \mathbf{p}_K\}$ is then recursively processed in the same way for the two line segments $\overline{\mathbf{p}_0, \mathbf{p}_m} \rightarrow \{\mathbf{p}_0, \mathbf{p}_l, \mathbf{p}_m\}$ and $\overline{\mathbf{p}_m, \mathbf{p}_K} \rightarrow \{\mathbf{p}_m, \mathbf{p}_r, \mathbf{p}_K\}$. This recursive refinement process defines a binary hierarchy over all the points

$\mathbf{p}_1, \ldots \mathbf{p}_l, \ldots \mathbf{p}_m \ldots \mathbf{p}_r \ldots \mathbf{p}_{K-1}$ (see also Fig. 5.8), i.e. with root $\mathbf{p}_m$ with left and right child nodes $\mathbf{p}_l$ and $\mathbf{p}_r$ respectively.

As illustrated in Fig. 5.7, the key idea of our approach is to apply the Douglas-Peucker refinement principle on the control polygon of a trail's B-spline and use the convex hull property mentioned in Sec. 5.3.2 to define a sequence of point insertions. Hence for a trail $P_i = \{\mathbf{p}_0, \ldots \mathbf{p}_K\}$ we start with its initial minimal control points $\mathbf{p}_j^0 = \widetilde{\mathbf{p}}_{i_{\text{first}}}, \ldots \widetilde{\mathbf{p}}_{i_{\text{last}}}$ (i.e. points $\mathbf{p}_1, \mathbf{p}_3, \mathbf{p}_{11}$ in Fig. 5.7(a)) implied by the reduced reference graph $\widetilde{\mathbf{R}}$. Starting with $l = 0$, all triangles $\triangle_j^l = \{\mathbf{p}_{j-1}^l, \mathbf{p}_j^l, \mathbf{p}_{j+1}^l\}$ are tested for intersection with any 3D obstacle or other interference constraints (see Fig. 5.7(b)). If an intersection for $\triangle_j^l$ is detected, its two line segments $\overline{\mathbf{p}_{j-1}^l, \mathbf{p}_j^l}$ and $\overline{\mathbf{p}_j^l, \mathbf{p}_{j+1}^l}$ are subdivided by points from $\mathbf{R}$ (i.e. inserting points $\mathbf{p}_2$ and $\mathbf{p}_6$ in Fig. 5.7(c)) to form a refined set of control points $\mathbf{p}_j^{l+1} \in \mathbf{R}$. This process is now recursively applied to the new triangles $\triangle_j^{l+1}$ (i.e. $\triangle_2$, $\triangle_3$ and $\triangle_6$ in Fig. 5.7(d)) until no more intersections are detected (e.g. $\triangle_2$ in Fig. 5.7(e)).

Variable LOD B-spline

The variable LOD B-spline model is based on the Douglas-Peucker refinement and a binary hierarchy over consecutive segments of control points. Note that each initial trail segment $\overline{\widetilde{\mathbf{p}}_j, \widetilde{\mathbf{p}}_{j+1}}$ and its subsequent Douglas-Peucker refinement as outlined above gives rise to a set of points added in between $\widetilde{\mathbf{p}}_j$ and $\widetilde{\mathbf{p}}_{j+1}$ which form a binary tree as illustrated in Fig. 5.8(a) which we refer to by $\widehat{\widetilde{\mathbf{p}}_j \widetilde{\mathbf{p}}_{j+1}}$. Hence each complex trail

$$P_i = \{\widetilde{\mathbf{p}}_0, \mathbf{p}_1, \ldots, \mathbf{p}_{s-1}, \widetilde{\mathbf{p}}_s, \mathbf{p}_{s+1}, \ldots \mathbf{p}_{t-1}, \widetilde{\mathbf{p}}_t, \mathbf{p}_{t+1}, \ldots \mathbf{p}_{K-1}, \widetilde{\mathbf{p}}_K\} \tag{5.2}$$

is now represented as a control polygon consisting of a sequence number of binary Douglas-Peucker refinement trees

$$\widehat{P}_i = \widehat{\widetilde{\mathbf{p}}_0 \widetilde{\mathbf{p}}_s}, \ \widehat{\widetilde{\mathbf{p}}_s \widetilde{\mathbf{p}}_t}, \ \ldots \widehat{\widetilde{\mathbf{p}}_k \widetilde{\mathbf{p}}_K}, \tag{5.3}$$

which together define a *hierarchical LOD control polygon structure* $\widehat{P}_i$ for the quadratic B-spline $s_i(u)$.

Given a certain LOD selection criterion that can be evaluated and stored in the control polygon structure $\widehat{P}_i$ and control point trees $\widehat{\widetilde{\mathbf{p}}_j \widetilde{\mathbf{p}}_{j+1}}$, it is possible to variably adjust the control polygon and thus the detail level of the displayed B-spline $s_i(u)$ accordingly. At the lowest detail level, $s_i(u)$ is defined by the minimal control polygon points of P_i only from $\widetilde{\mathbf{R}}$, i.e. only points $\widetilde{\mathbf{p}}_0, \widetilde{\mathbf{p}}_s, \widetilde{\mathbf{p}}_t, \ldots \widetilde{\mathbf{p}}_K$ in Eq. 5.2, while at the highest LOD the control polygon includes all points of P_i from $\mathbf{R}$. Fig. 5.8(b) illustrates a geometry-occlusion based control point selection in the

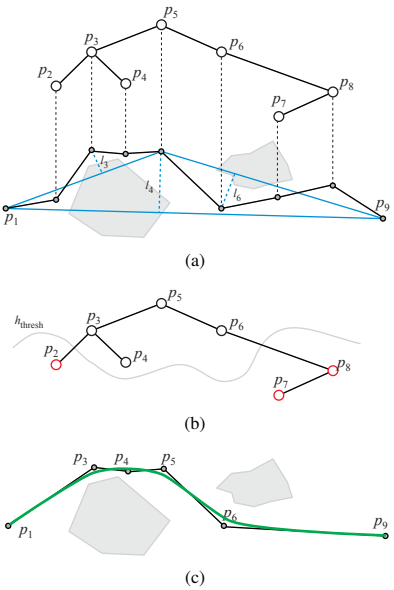


Figure 5.8: *Variable LOD B-spline representation. (a) Original control polygon around obstacles and Douglas-Peucker induced binary vertex hierarchy. (b) Example control point selection using terrain intersection information (front elevation view). (c) Resulting control polygon selection guaranteeing a non-intersecting B-spline around obstacles (top-down view).*

binary refinement tree giving rise to the control polygon in Fig. 5.8(c). The resulting B-spline will display a smooth curve following the reference network path P_i but avoiding intersection with the given scene geometry.

Given a 3D digital terrain elevation model as B-spline simplification constraints, we apply the proposed framework as follows. For each trail P_i given in a 3D terrain environment, we test the height field intersection constraints by issuing hardware accelerated occlusion queries for the control point triangles $\triangle_j^l$ on the 3D terrain obstacle at different elevations h above ground as shown in Fig. 5.9. Starting with a maximal height offset $h_{\max}$ for $l = 0$, we continuously reduce it for $l \rightarrow l+1$ (Fig. 5.9(c)) and record for every required control polygon refinement – adding new points $\mathbf{p}_{j\pm1}^{l+1}$ because $\triangle_j^l$ interferes with the 3D terrain – the height off-

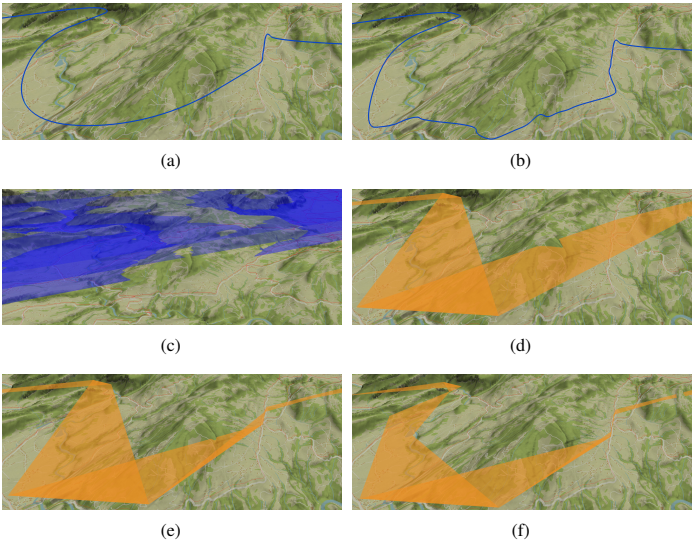


Figure 5.9: *The 3D B-spline refinement levels from a perspective view. (a) Coarsest B-spline LOD, at high altitude offset, induced by reduced reference network graph. (b) Refined B-spline LOD at lower height offset avoiding terrain intersections. (c) Different height offsets (blue planes) at which 3D obstacle interferences are tested with respect to the terrain elevation model. (d,e,f) Quadratic B-spline refinements and control-point triangle occlusion queries evaluated against the 3D terrain at different height offsets.*

set at which the intersection occurs. Figs. 5.9(d) to 5.9(f) show the control point triangle intersection queries with the terrain at different height offsets. Thus for each trail segment tree $\widetilde{\mathbf{p}_s\mathbf{p}_t}$, eventually all inner points $\mathbf{p}_j = \mathbf{p}_{s+1} \ldots \mathbf{p}_{t-1}$ record their height offset h_j at which they must be included for B-spline refinement to avoid intersection with the 3D terrain environment. At rendering time, any variable LOD B-spline can thus be created by an inorder traversal of the tree segments $\widetilde{\mathbf{p}_s\mathbf{p}_t}$ of the control polygon hierarchy $\widehat{P}_i$, and by selecting all control points $\mathbf{p}_j$ for which their height offset h_j is above a given threshold h_{thresh}. Variable LOD B-spline examples at different height offsets are rendering in Figs. 5.9(a), 5.9(b) and 5.10.

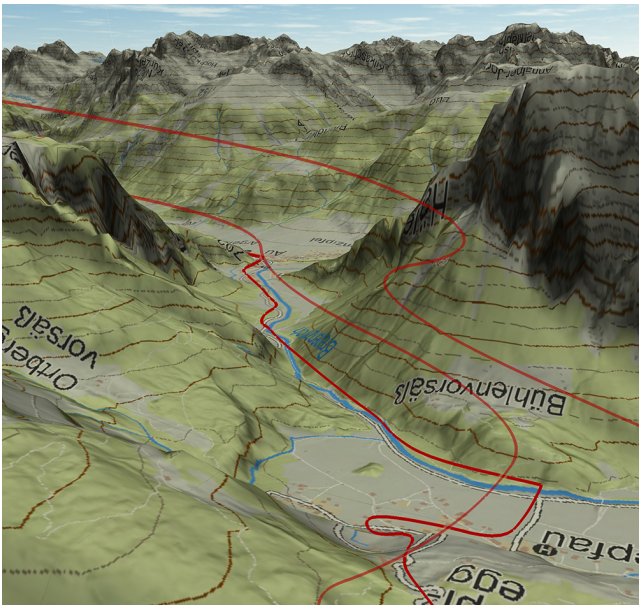


Figure 5.10: *Screenshot showing different B-spline LODs at different height offsets over the 3D terrain.*

5.4 Interactive Constrained Graph Bundle Visualization

Given the input graph $\mathbf{G}$, reference vector map network $\mathbf{R}$ and a 3D digital elevation model, the preprocess as outlined in Sec. 5.3 and illustrated in Fig. 5.5 generates our proposed reference-network and terrain-height-field constrained variable LOD B-spline representations $\widehat{P}_i$ for all given point-to-point connections c_i.

At run-time, the visualization application's data preparation tasks are to select the B-splines control point trees $\widehat{P}_i$ of all paths P_i to be displayed, and to generate the variable LOD B-spline control polygon from the points $\mathbf{p}_j \in \widehat{P}_i$ for which $h_j > h_{\text{tresh}}$ for a user given threshold h_{tresh}, i.e. height offset above ground. A drawable representation for each $\widehat{P}_i$ is then generated as an OpenGL 3D line-strip

Data sets	**Vorarlberg:**	**Switzerland small:**	**Switzerland:**	**US Migration:**	**World Airlines:**
G					
$\|L\|$	105	2886	2886	3235	3457
$\|C\|$	3442	17125	31608	387841	68382
R					
$\|V\|$	9258	975491	975491	65761	1778
$\|E\|$	83017	902525	902525	65177	1084
Graph reduction	0.1s	2.9s	2.9s	0.01s	0.01s
$\widetilde{\mathbf{R}}$					
$\|\widetilde{V}\|$	598	36736	36736	785	395
$\|\widetilde{E}\|$	4586	55033	55033	1072	798
Startpoint queries	0.1s	1.8s	2.5s	8.7s	4.4s
Shortest path	0.4s	225.8s	424.5s	51.3s	35.4s
Line hierarchy	4.5s	-	443.7s	102.3s	6.6s
FDEB	153s	1-2 h	-	-	-

Table 5.1: *Cluttered graph (***G***) and reference network (***R***) data sizes as well as processing times for reference graph reduction, nearest point shortest path queries, control point hierarchy generation and the force-directed edge bundling calculation. Due to memory limits, FDEB timings are available only for smaller graphs. Experiments were conducted on a system implemented in C++ and OpenGL using an Intel Core i7-3770K 3.50GHz, 16GB RAM, Geforce 680 GTX with 4GB RAM, and Windows 7.*

by evaluating $s_i(u)$ for small parameter intervals Δu. Alternatively, a De Boor based subdivision algorithm could be used to refine the control polygon such as to eventually use it as an OpenGL line strip primitive. Note that this processing step only has to be done once for each displayed trail when the height offset threshold h_{tresh} changes.

Given all the B-splines to-be displayed as OpenGL 3D line strips, they can be rendered together with the 3D scene environment using standard 3D rendering techniques for surfaces, e.g. the grid digital terrain height field, and for line segments, i.e. the B-spline line strips. For a high quality result, our real-time rendering of the 3D terrain and B-spline line strips exploits a deferred shading pipeline using multiple framebuffer objects and stencil information.

The B-spline line-strip objects are rendered into a stencil-framebuffer such as to enable stencil based colorization and other more complex image processing operations. This allows image-based manipulations of the rendered B-spline curves before and together with the final compositing with the 3D scene environment. Currently the stencil is exploited for evaluating the bundling intensity by recording for each pixel how many B-splines contribute to it.

Besides the stencil based line strip rendering, all other 3D scene objects are rendered in one main pass into a separate buffer and combined with the line strip. A final composition stage merges all framebuffer information and delivers a super-sampled high resolution image for subsequent downsampling to the required screen resolution. We use a doubled resolution of all framebuffers to achieve a nice anti-aliased image in the end.

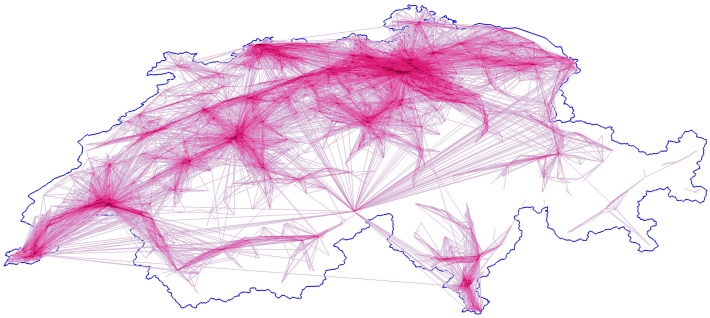
5.5 Results

Our application allows to start and visualize the preprocessing phase for any combination of datasets. In particular, it supports exploration and visualization of the variable LOD reference-network bundled and 3D terrain-intersection constrained graphs in a full 3D environment at interactive frame rates. The current implementation is not optimized for performance but for the realization of the full functionality of the proposed constrained bundling framework.

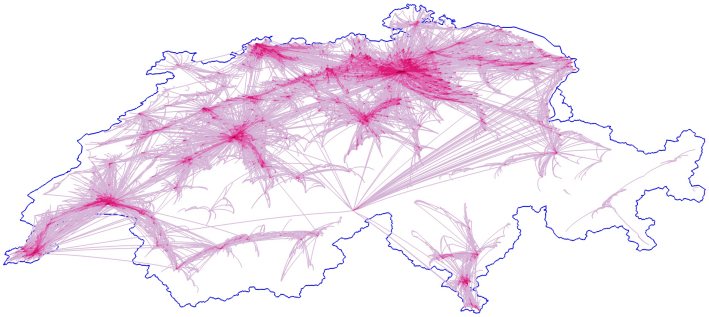
Technical details as well as the computational cost of our reference bundling (REFB) can be found in Tab. 5.1. Our REFB method depends primarily on the input data complexity and has not yet been optimized for performance. Nevertheless, in comparison to FDEB (Fig. 5.11(b)) our approach scales better, primarily due to FDEB's quadratic memory and high computational cost requirements in the number of edges.

As shown in Fig. 5.12(b), the bundled commuter data makes it possible to identify which routes are used the most and which highways or main roads are most likely impacted from the commuter traffic. The comparison in Fig. 5.11(b) shows that this can only be achieved if the bundling is constrained to some underlying and routing network information following hard constraints. The visual effect is similar to a street colorization as in Fig. 5.12(a). However the B-spline interpolation is smoother and shows more generalized paths which reduces the distraction for the user and emphasizes routing highlights better. The B-spline representation can also indicate the uncertainty of the applied (shortest path) routing in a visual sense. In comparison to a road colorization, the visual interpretation is not tied to an exact road. We argue that using a B-spline representation for transporting the information of uncertainty is preferable over a direct road colorization to identify highlights.

In addition to the comparison above, our method allows the integration of the bundled result in a three-dimensional scene and can take 3D graph and 3D vector map data into account but also supports 3D obstacle avoidance. As shown in Figs. 5.1, 5.9, 5.10, 5.16(a) and 5.16(b) bundling results can be displayed as variable LOD B-spline curves correctly in a 3D environment, thus supporting correct perspective viewing and depth-occlusions. The bundled variable LOD B-splines



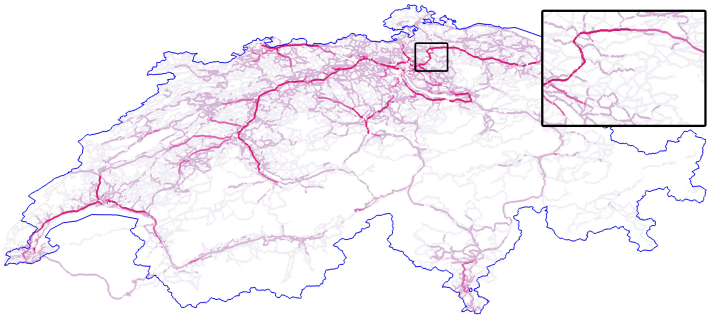
(a) Original data



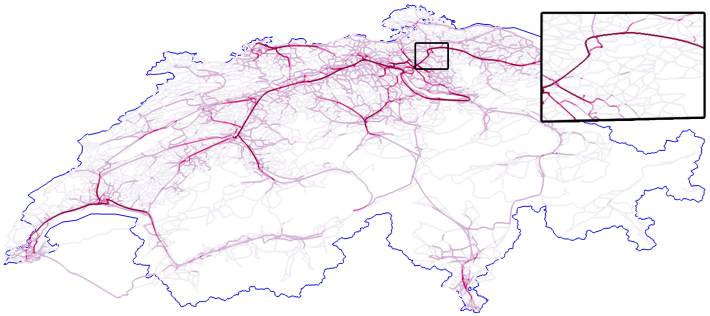
(b) FDEB

Figure 5.11: *Visual results of the Swiss commuter data. Strong colors indicate overlapping commuter streams. (a) Raw dataset containing the home-to-work travel information as town-to-town connections. (b) FDEB bundled graph.*

can be rendered directly into a 3D scene environment and therefore integration of final image composition methods in a perspective view with complex depth-occlusions are possible. In the most detailed LOD resolution the visual effect is similar to a street colorization. But for coarser detail levels, the adaptive B-spline LOD hierarchy can be exploited, and coupled with or used as indication of the uncertainty of the applied (shortest path) routing over the reference network. This allows to interactively change the geometry of path bundles based on an expected



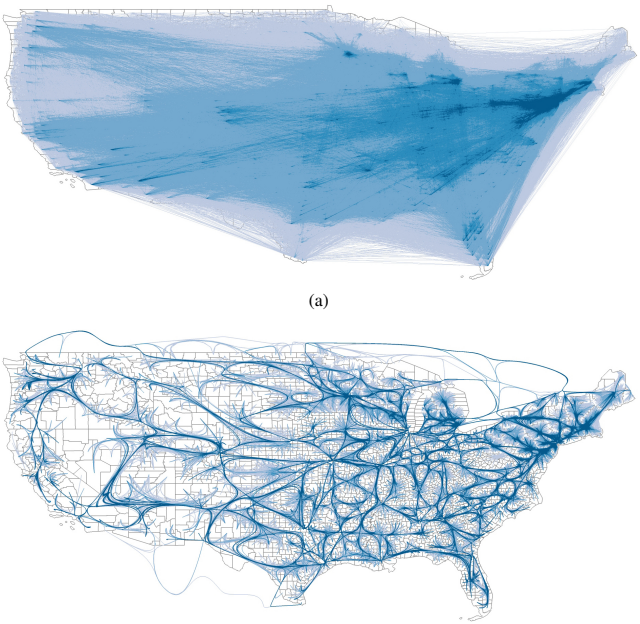
(a) Road Colorisation



(b) REFB

Figure 5.12: *Second part of visual results of the Swiss commuter data. (c) Colored street data set. (d) Vector map constrained path bundling.*

uncertainty value or other LOD attribute. If there is low evidence for commuter traffic following a specific route, path bundles can visualize this uncertainty in a geometric sense at lower LOD. This is based on the assumption that the geometric accuracy of the bundled paths following the constraint network routes improves the uncertainty information visually than using direct road colorization where this information does not appear.



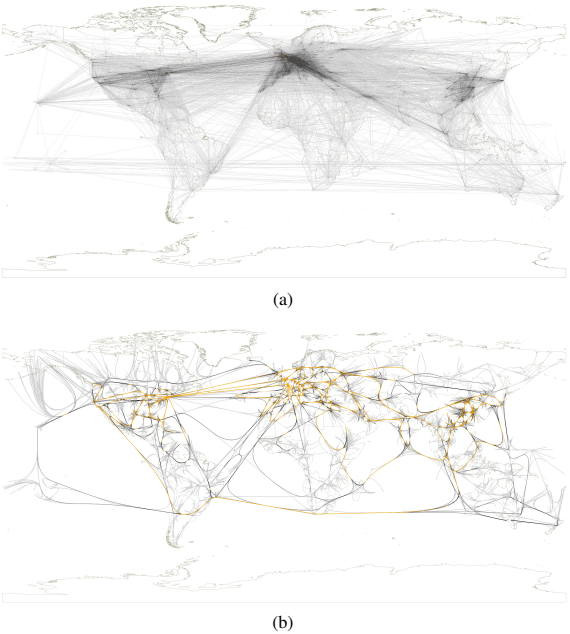
(a)

(b)

Figure 5.13: *(a) The raw US migration dataset containing county-to-county migration information. (b) The bundled data set using the US train line data as reference network.*

5.6 Conclusions and Future Work

In this chapter we present a novel approach for vector-map constrained path bundling for 2D and 3D environments. Furthermore, we describe a hierarchical multiresolution control polygon structure that cannot only be used for rendering variable LOD B-splines, but which in particular can incorporate obstacle interference and intersection constraints in a 3D environment. Additionally we present a hardware occlusion-query based method for detecting intersections of (quadratic) B-spline segments with 3D obstacles. The examples shown in the results are mainly from geospatial data sets, but our method is not domain specific to GIS systems. However, it requires and is targeted at a pair of co-registered (dense) input graph and



(a)

(b)

Figure 5.14: *The world airline dataset (a) raw flight connections. (b) bundled over a virtual air corridor connection network with color encoded flight intensity (orange for heavier air traffic).*

reference network datasets embedded in a 2D or 3D environment. Our method is not aimed at general graphs or other simpler standard graphs which are sometimes used for testing in recent related work (e.g. as in [Ersoy et al., 2011; Hurter et al., 2012]). Nevertheless, bundling general graphs could be achieved by providing an artificial reference network or generating a reference network from the raw graph data itself, e.g. using the minimum spanning tree of the graph. In fact, an artificial reference network is shown in Fig. 5.14 for the world airline graph data, where the reference network of air corridors was modeled by cartographic domain experts according to their needs. In general, lots of reference networks exist in many domains, on top of which additional graph based relational information can be

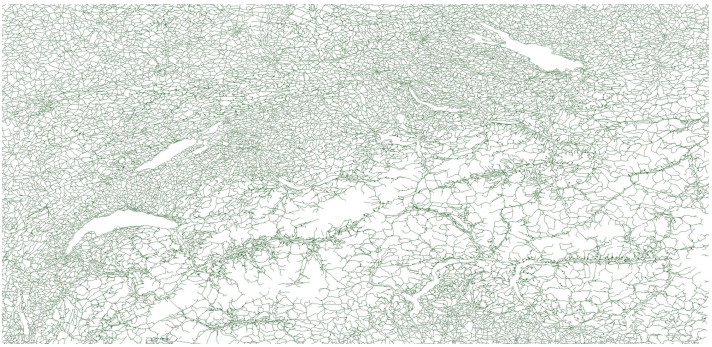
analyzed and visualized.

The major limitation of our method is the non-optimized implementation. For handling very large input graphs and reference networks of many 10s or 100s of thousands of connections, the variable LOD B-spline generation has to be improved in its use of hardware occlusion queries. Moreover, for large graphs the resulting massive amounts of line-strip geometry for the bundled B-splines cause limitations in interactivity. Thus only a subset of bundled B-splines can be explored interactively. These limitations are the main targets for future work. Further algorithmic improvements may be applicable to reference graph reduction, shortest path finding or routing, but these would primarily improve preprocessing but not the visual results. Furthermore, the precision of occlusion queries is limited. Occlusion queries depend on the viewport size and projection parameters, which may potentially lead to errors in the B-spline refinement procedure. Due to the similarity of this occlusion detection method to shadow maps, similar strategies could be applied such as multiple viewport snapshots to improve the data precision.

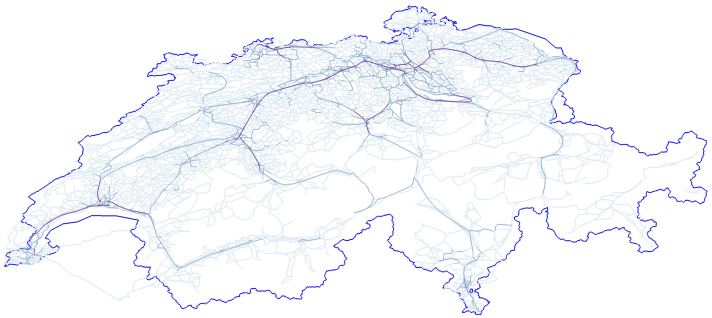
Thus the main limitations revolve around performance issues for handling massive vector map data sets. The performance of the variable LOD B-spline refinement has to be improved significantly as well as online line-strip generation and real-time rendering of massive amounts of line geometry.

5.6.1 Streaming Applications

The assumption that data sets are complete and available before the bundling precludes potential path-bundling visualizations in a streaming environment. Our system has the ability to be used in a data streaming environment as well. The point-to-point data sets do not need to be known and complete beforehand. Our approach offers the possibility to establish an on-demand service, for example for daily commuter travelers, daily taxi tours or other higher frequency connection data, where the connection stream is processed incrementally as connections are sequentially reported. This also means that one could maintain a time-window of actual connection data and bundled paths for visualization, drop old data and update it with newer connections based on the recorded time. By only taking a static reference network into account, a single path result will be consistent as if the method were applied on the entire data set at once, independent of how many bundles are processed and shown. This is a significant difference to approaches such as the FDEB method. The FDEB approach uses a precomputed measurement matrix storing relation weightings of all connections and calculates attraction forces between connections so that every point-to-point connection influences the final result. Therefore, in contrast to our method, the whole visualization can change with a single new connection and is thus not streaming-capable.



(a)



(b)

Figure 5.15: *The results for Swiss commuter dataset in Fig. 5.11(a). (a) an example for a reference network showing major road data of Switzerland (1:200'000). (b) constrained bundling of commuter paths highlighting the major traffic axis visually as hot spots.*

(a)

(b)

Figure 5.16: *Second part of the results for Swiss commuter dataset in Fig. 5.11(a). (c) A zoom-in view of the highly frequented larger area of Zurich. (d) A perspective view snapshot including terrain context with edge highlights.*

CHAPTER 6

CONCLUSIONS

6.1 Summary

This dissertation presents a framework for geographical data visualization. It is a software product subsequent to the research efforts in the VMML group over the past several years and projects in terrain rendering and geographical visualization. The *GlobeEngine* framework provides students and scientific personnel the ability for faster prototyping of visualization software and it also shows several algorithmic implementations in a complex geographical application comparable with industrial products.

One aspect of this thesis is the construction of a terrain rendering engine based on prior work done in the VMML group. We adapt existing algorithms and build up a completely new framework adapted to nowadays requirements. In the course of this adaption, data structures of the *RASTeR* algorithm have to be updated and a multipass rendering pipeline is introduced. Also this thesis shows highlights of new capabilities of the *RASTeR* terrain rendering system. The terrain rendering system has to be seen as an ongoing process of development and refactoring, because it can be the basis for further research in terrain visualization. The *GlobeEngine* provides on one hand the flexibility to easily extend and adapt terrain visualizations and on the other hand it offers an existing set of algorithms for terrain visualizations for state-of-the-art comparisons of future implementations of terrain algorithms.

The *GlobeEngine* terrain visualization is also the basis for several contribu-

tions in this thesis in the field of vector map rendering. Interesting challenges with vector maps often go hand in hand with a perspective terrain visualization. Therefore, an existing terrain visualization system is required for comparisons. One of the contributions of this thesis is a new technique to render large-scale vector maps on an existing terrain visualization with the pixel accuracy. Furthermore, deferred vector maps allow new customization and interaction possibilities for end users of the geographical visualization such as vector map blending or interactive advanced vector map styling. This thesis also shows that rendering general geographic features with post processing methods so called deferred rendering engines have the potential to replace traditional rendering pipelines and that in the future customized deferred rendering pipelines will be more important.

The thesis also contributes to the field of visual analytics by providing geographically relevant graph bundling algorithm. Terrain rendering and vector map renderings are core elements for the visualization of traffic information and also a relevant part when applying graph bundling algorithms to traffic data sets. The thesis shows a new way to implement a graph bundling algorithm based on a reference network. The method reduces cluttering by bundling graph edges and guide this streams related to existing traffic information in a perspective $3D$ scene. We also implemented a reference algorithm and extended this algorithm in $3D$ to compare the algorithms in terms of performance and visual results.

6.2 Future Work

The *GlobeEngine* system and the techniques presented in this thesis solve specific problems within a virtual globe or a general geographic information application. During the development of this work, new challenges opened up which will be described in the following paragraphs. The following challenges can be partially found in [Thöny et al., 2015]:

- **Terrain Visualization:** It can be said that the geometric part of terrain rendering is a solved problem. Current terrain visualization systems provide already a variety of algorithms to solve the triangulation and rendering of a terrain system in a good way. Furthermore, there are approaches for modeling, editing and sculpting of terrains. There is also a variety of software available for these use cases. However, there are new upcoming use cases in terrain visualization as the incorporation of point cloud data, uncertainty visualization on terrains and frameworks for the integration of a full $3D$ terrain data structure preserving the positive aspect from $2,5D$ height fields.

  The incorporation of $3D$ point cloud systems into existing terrain rendering system will be an important future challenge. $3D$ scanners, optical sensors

or *LiDAR* (Light detection and ranging) devices cause already a huge growth in available data. A lot of these systems will be used for real-time measurement and registration of objects to make software more intelligent, such as traffic management systems. It has to be assumed that the data growth goes on. Thus, terrain visualization systems have to display large amounts of terrain and point cloud data on embedded hardware as well, e.g. in cars. It also means that terrain visualization is shifting from $2,5D$ heightfield rendering to a full $3D$ problem, because the accumulated point clouds are not necessarily equal to $2,5D$ height fields. This will require the use of new data structures, for example sparse voxel octrees to combine terrain data and point cloud data in a unified way.

Another direction of research will be the rendering of terrain with volume based approaches. As already mentioned in Chapter 3, there are methods for rendering terrain which are based on raycasting algorithms. Raycasting is mainly used for volume rendering and a very important application are climate data sets. Obviously, climate data sets are spatially connected to terrain visualization. This means, a visualization using terrain data and climate data might have advantages from a unified rendering system only consisting of raycasting algorithms in opposite to a hybrid triangle & raycasting pipeline. However, it is clear that raycasting algorithms will be restricted to domain user desktop applications, because the hardware requirements for mobile and embedded graphics cards are still too exacting.

Furthermore, terrain rendering algorithms are challenged again by planetary scale rendering and planetary exploration. Terrain visualization data is still limited to data from Earth and recently from Mars. However, with growing precision of telescopes, satellite and drones, we will also have the possibility to explore other planets and asteroids. There are many more objects in our solar system which we will be able to scan and explore in the near future. Terrain rendering will be the core tool for the visualization and exploration of such data.

- **Deferred Vector Maps** Vector map visualizations produce the biggest amount of information in geographic information systems nowadays. A lot of applications such as navigation systems and map software already depend on vector map rendering. Vector maps cause a lot of rendering problems, because a lot of flexibility inside the visualizations are required. An engine has to support a variety of visualization strategies to make vector maps a useful tool for communication. Tools like Mapbox[1] try to make the design

[1] https://www.mapbox.com/

of vector maps easier. However, this improves the creation of maps but not the creation of new kinds of visualizations.

The *GlobeEngine* system will further develop into the topic of large-scale rendering. In addition to the large-scale line rendering system presented in Chapter 4, polygon based data sets are an important topic as well as multiresolution hierarchies for loading parts of data sets. Furthermore, the technique of deferred screen space vector maps can be improved by perspective corrections. Line and polygon data often appear much bigger due to distortions caused by mapping a $2D$ shape on a $3D$ surface with a parallel projection. With the deferred vector map approach it is possible to correct these distortions while rendering by adjusting vertices and pixels according to the line or polygon information.

Furthermore, the *GlobeEngine* has the goal to support vector maps also as more complex objects. Terrain visualization can be seen as information visualization but also as an experience. From an information visualization perspective, it makes sense to render geographic objects, for example a lake in a simple form with name and ferry routes. However, from a point of exploration and experience it would make more sense to present a lake with a realistic water rendering, animated with waves for example. It makes the exploration process more interesting but it is highly dependent on the target user group. However, such visualizations will also need a group of designers or redactors creating such content and a flexible system to provide functionality to integrate this content. Various technical challenges can appear, such as illumination challenges for abstract representations of housing, farming ground, lakes and snow cover, comic or illustrative renderings challenges or physics based simulations such as mud or snow avalanches, weather simulation or wave propagation.

- **Graph bundling within perspective 3D:** Graph bundling already evolved to a standard tool for graph visualizations. Usually graphs are abstract forms of data visualization. When connected to spatial data, graphs can give a lot of insight into data sets and can visualize spatial connections easily. Traffic and migration are two current topics which will be able to profit from spatial graph visualization. An active topic in graph bundling is the search for quality metrics for graph bundling methods. Until now, the research is user and application centric, but in the near future, there will be more metrics coming up to diversify bundling methods better.

  From a technical point of view graph bundling is a nice topic to try new mathematical methods. A future topic to improve the $3D$ perspective graph bundling is the usage of an electric field bundling which is repelling bundles

away from mountains. The force information can be calculated from height fields and normal vectors. At first sight, bundles would automatically go around mountains and also obstacles. Another aspect are the bundles themselves. The rendering of the bundles in this work is using traditional line rendering but there are topics to improve the visual quality of lines, such as illuminated lines or opacity correction of lines.

I am certain, that the topic of large-scale geographic visualization will deliver a lot of interesting challenges in the future in terms of computational geometry, rendering algorithms or intelligent systems. However, the most important principle for visualizations remains the same, namely the questions, what is visualized, why is it visualized and how is it visualized.

BIBLIOGRAPHY

[Akenine-Moller et al., 2002] Akenine-Moller, T., Moller, T., and Haines, E. (2002). *Real-Time Rendering*. A. K. Peters, Ltd.

[Andersson, 2007] Andersson, J. (2007). Terrain rendering in frostbite using procedural shader splatting. In *ACM SIGGRAPH 2007 Courses*, pages 38–58.

[Baboud et al., 2012] Baboud, L., Eisemann, E., and Seidel, H.-P. (2012). Precomputed safety shapes for efficient and accurate height-field rendering. *IEEE Transactions on Visualization and Computer Graphics*, 18(11):1811–1823.

[Bavoil and Sainz, 2008] Bavoil, L. and Sainz, M. (2008). Screen space ambient occlusion. In *ShaderX 7*. NVIDIA Corporation.

[Bettio et al., 2007] Bettio, F., Gobbetti, E., Marton, F., and Pintore, G. (2007). High-quality networked terrain rendering from compressed bitstreams. In *Proceedings International Conference on 3D Web Technology*, pages 37–44.

[Bösch et al., 2009] Bösch, J., Goswami, P., and Pajarola, R. (2009). RASTeR: Simple and efficient terrain rendering on the GPU. In *Proceedings Eurographics - Area Papers*, pages 35–42.

[Bruneton and Neyret, 2008] Bruneton, E. and Neyret, F. (2008). Real-time rendering and editing of vector-based terrains. *Computer Graphics Forum*, 27(2):311–320.

[Caserta et al., 2011] Caserta, P., Zendra, O., and Bodenes, D. (2011). 3D hierarchical edge bundles to visualize relations in a software city metaphor. In *Proceedings IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 1–8.

[Chajdas et al., 2011] Chajdas, M. G., Mcguire, M., and Luebke, D. (2011). Subpixel reconstruction antialiasing for deferred shading. *Proceedings Interactive 3D Graphics and Games*, pages 15–22.

[Christen, 2008] Christen, M. (2008). The future of virtual globes - the interactive ray-traced digital earth. In *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, volume XXXVII-B2, pages 969–974.

[Clasen and Hege, 2006] Clasen, M. and Hege, H.-C. (2006). Terrain rendering using spherical clipmaps. In *Proceedings Eurographics / IEEE VGTC Symposium on Visualization*, pages 91–98.

[Cozzi and Ring, 2011] Cozzi, P. and Ring, K. (2011). *3D Engine Design for Virtual Globes*. A. K. Peters, Ltd.

[Cui et al., 2008] Cui, W., Zhou, H., Qu, H., Wong, P. C., and Li, X. (2008). Geometry-based edge clustering for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1277 –1284.

[Dai et al., 2008] Dai, C., Zhang, Y., and Yang, J. (2008). Rendering 3D vector data using the theory of stencil shadow volumes. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 37:643–648.

[Delling et al., 2011] Delling, D., Goldberg, A. V., Nowatzyk, A., and Werneck, R. F. (2011). PHAST: Hardware-accelerated shortest path trees. In *Proceedings IEEE International Parallel & Distributed Processing*, pages 921–931.

[Deng et al., 2013] Deng, B., Xu, D., Zhang, J., and Song, C. (2013). Visualization of vector data on global scale terrain. In *Proceedings International Conference on Computer Science and Electronics Engineering*, pages 85–88.

[Dick et al., 2009a] Dick, C., Krüger, J., and Westermann, R. (2009a). GPU raycasting for scalable terrain rendering. In *Proceedings Eurographics - Area Papers*, pages 43–50.

[Dick et al., 2010] Dick, C., Krüger, J., and Westermann, R. (2010). GPU - aware hybrid terrain rendering. In *Proceedings of IADIS Computer Graphics, Visualization, Computer Vision and Image Processing*, pages 3–10.

[Dick et al., 2009b] Dick, C., Schneider, J., and Westermann, R. (2009b). Efficient geometry compression for GPU-based decoding in realtime terrain rendering. *Computer Graphics Forum*, 28(1):67–83.

[Dimitrijević and Rančić, 2015] Dimitrijević, A. M. and Rančić, D. D. (2015). Ellipsoidal clipmaps - a planet-sized terrain rendering algorithm. *Computers and Graphics*, 52(C):43–61.

[dos Santos and Brodlie, 2004] dos Santos, S. and Brodlie, K. (2004). Gaining understanding of multivariate and multidimensional data through visualization. *Computers and Graphics*, 28(3):311–325.

[Durdevic and Tartalja, 2013] Durdevic, D. M. and Tartalja, I. I. (2013). Hfpac: Gpu friendly height field parallel compression. *Geoinformatica*, 17(1):207–233.

[Eilemann et al., 2009] Eilemann, S., Makhinya, M., and Pajarola, R. (2009). Equalizer a scalable parallel rendering framework. *IEEE Transactions on Visualization and Computer Graphics*, 15(3):436–452.

[Ellis and Dix, 2007] Ellis, G. and Dix, A. (2007). A taxonomy of clutter reduction for information visualisation. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1216–1223.

[Ersoy et al., 2011] Ersoy, O., Hurter, C., Paulovich, F., Cantareiro, G., and Telea, A. (2011). Skeleton-based edge bundling for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2364 –2373.

[Gansner et al., 2011] Gansner, E., Hu, Y., North, S., and Scheidegger, C. (2011). Multilevel agglomerative edge bundling for visualizing large graphs. In *Proceedings IEEE Pacific Visualization Symposium*, pages 187 –194.

[Gerstner, 2003] Gerstner, T. (2003). Top-down view-dependent terrain triangulation using the octagon metric. In *Proceedings Eurographics Symposium on Geometry Processing*, pages 1–11.

[Goldberg and Harrelson, 2005] Goldberg, A. V. and Harrelson, C. (2005). Computing the shortest path: A search meets graph theory. In *Proceedings ACM-SIAM Symposium on Discrete algorithms*, pages 156–165.

[Goswami et al., 2010] Goswami, P., Makhinya, M., Bösch, J., and Pajarola, R. (2010). Scalable parallel out-of-core terrain rendering. In *Proceedings Eurographics Symposium on Parallel Graphics and Visualization*, pages 63–71.

[Gross and Pfister, 2007] Gross, M. and Pfister, H. (2007). *Point-Based Graphics*. Morgan Kaufmann Publishers Inc.

[Guo, 2009] Guo, D. (2009). Flow mapping and multivariate visualization of large spatial interaction data. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1041 –1048.

[Hildebrandt and Döllner, 2010] Hildebrandt, D. and Döllner, J. (2010). Service-oriented, standards-based 3d geovisualization: Potential and challenges. *Computers, Environment and Urban Systems*, 34(6):484–495.

[Hoberock and Jia, 2007] Hoberock, J. and Jia, Y. (2007). High-quality ambient occlusion. In *GPU Gems 3*, pages 257–274. Addison-Wesley Professional.

[Holten, 2006] Holten, D. (2006). Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):741 –748.

[Holten and van Wijk, 2009] Holten, D. and van Wijk, J. J. (2009). Force-directed edge bundling for graph visualization. *Computer Graphics Forum*, 28(3):983–990.

[Hurter et al., 2013a] Hurter, C., Ersoy, O., Fabrikant, S. I., Klein, T. R., and Telea, A. C. (2013a). Bundled visualization of dynamic graph and trail data. *IEEE Transactions on Visualization and Computer Graphics*, 99:1.

[Hurter et al., 2012] Hurter, C., Ersoy, O., and Telea, A. (2012). Graph bundling by kernel density estimation. *Computer Graphics Forum*, 31:865–874.

[Hurter et al., 2013b] Hurter, C., Ersoy, O., and Telea, A. (2013b). Smooth bundling of large streaming and sequence graphs. In *Proceedings IEEE Pacific Visualization Symposium*, pages 41–48.

[Kersting and Döllner, 2002] Kersting, O. and Döllner, J. (2002). Interactive 3D visualization of vector data in GIS. In *Proceedings ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 107–112.

[Kobbelt and Botsch, 2004] Kobbelt, L. and Botsch, M. (2004). A survey of point-based techniques in computer graphics. *Computers and Graphics*, 28(6):801–814.

[Kooima et al., 2009] Kooima, R., Leigh, J., Johnson, A. E., Roberts, D., SubbaRao, M., and DeFanti, T. A. (2009). Planetary-scale terrain composition. *IEEE Transactions on Visualization and Computer Graphics*, 15(5):719–733.

[Koonce, 2007] Koonce, R. (2007). Deferred shading in tabula rasa. In *GPU Gems 3*, pages 7–37. Addison-Wesley Professional.

[Lambers and Kolb, 2012] Lambers, M. and Kolb, A. (2012). Ellipsoidal cube maps for accurate rendering of planetary-scale terrain data. In *Proceedings Pacific Graphics*.

[Lambert et al., 2010a] Lambert, A., Auber, D., and Melançon, G. (2010a). Living flows: Enhanced exploration of edge-bundled graphs based on gpu-intensive edge rendering. In *Proceedings International Conference Information Visualisation*, pages 523 –530.

[Lambert et al., 2010b] Lambert, A., Bourqui, R., and Auber, D. (2010b). 3D edge bundling for geographical data visualization. In *Proceedings International Conference Information Visualisation*, pages 329–335.

[Lambert et al., 2010c] Lambert, A., Bourqui, R., and Auber, D. (2010c). Winding roads: Routing edges into bundles. *Computer Graphics Forum*, 29(3):853–862.

[Lee et al., 2006] Lee, B., Plaisant, C., Parr, C. S., Fekete, J.-D., and Henry, N. (2006). Task taxonomy for graph visualization. In *Proceedings AVI Workshop on Beyond Time and Errors: Novel Evaluation Methods for Information Visualization*, pages 1–5.

[Li et al., 2005] Li, Z., Zhu, Q., and Gold, C. (2005). *Digital Terrain Modelling*. CRC Press.

[Liktor and Dachsbacher, 2012] Liktor, G. and Dachsbacher, C. (2012). Decoupled deferred shading for hardware rasterization. In *Proceedings ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 143–150.

[Lim and Choi, 2008] Lim, C.-G. and Choi, B. (2008). Hierarchical triangular patches for terrain rendering with their matching blocks. In *Proceedings International Conference on Digital Interactive Media in Entertainment and Arts*, pages 327–334.

[Livny et al., 2009] Livny, Y., Kogan, Z., and El-Sana, J. (2009). Seamless patches for GPU-based terrain rendering. *The Visual Computer*, 25(3):97–208.

[Luo et al., 2012] Luo, S.-J., Liu, C.-L., Chen, B.-Y., and Ma, K.-L. (2012). Ambiguity-free edge-bundling for interactive graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 18(5):810 –821.

[Mahdavi-Amiri et al., 2015] Mahdavi-Amiri, A., Alderson, T., and Samavati, F. (2015). A survey of digital earth. *Computers and Graphics*, 53:95–117.

[Munkberg et al., 2006] Munkberg, J., Akenine-Möller, T., and Ström, J. (2006). High quality normal map compression. In *Proceedings ACM SIGGRAPH/Eurographics Symposium on Graphics Hardware*, pages 95–102.

[Munzner, 2014] Munzner, T. (2014). *Visualization Analysis and Design*. A. K. Peters, Ltd.

[Ohlarik and Cozzi, 2011] Ohlarik, D. and Cozzi, P. (2011). A screen-space approach to rendering polylines on terrain. In *ACM SIGGRAPH Posters*, pages 68:1–1.

[Olsson et al., 2012] Olsson, O., Billeter, M., and Assarsson, U. (2012). Clustered deferred and forward shading. In *Proceedings ACM SIGGRAPH/Eurographics Symposium on High-Performance Graphics*, pages 87–96.

[Pajarola, 1998] Pajarola, R. (1998). Large scale terrain visualization using the restricted quadtree triangulation. In *Proceedings IEEE Visualization*, pages 19–26,515.

[Pajarola and Gobbetti, 2007] Pajarola, R. and Gobbetti, E. (2007). Survey of semi-regular multiresolution models for interactive terrain rendering. *Visual Computing*, 23(8):583–605.

[Pérez et al., 2004] Pérez, M., Olanda, R., and Fernández, M. (2004). Visualization of large terrain using non-restricted quadtree triangulations. In *Proceedings ICCSA International Conference Computational Science and Its Applications Part II*, pages 671–681.

[Persson, 2012] Persson, E. (2012). Graphics gems for games: Findings from avalanche studios. ACM SIGGRAPH Advances in Real-Time Rendering in Games - Course Material.

[Ripolles et al., 2012] Ripolles, O., Ramos, F., Puig-Centelles, A., and Chover, M. (2012). Real-time tessellation of terrain on graphics hardware. *Computers & Geosciences*, 41:147–155.

[Sainz et al., 2004] Sainz, M., Pajarola, R., and Lario, R. (2004). Points reloaded: Point-based rendering revisited. In *Proceedings Eurographics Symposium on Point-Based Graphics*, pages 121–128.

[Saito and Takahashi, 1990] Saito, T. and Takahashi, T. (1990). Comprehensible rendering of 3-D shapes. In *Proceedings ACM SIGGRAPH*, pages 197–206.

[Samet, 2005] Samet, H. (2005). *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc.

[Sample and Ioup, 2010] Sample, J. T. and Ioup, E. (2010). *Tile-Based Geospatial Information Systems: Principles and Practices*. Springer-Verlag.

[Schneider and Westermann, 2006] Schneider, J. and Westermann, R. (2006). GPU-friendly high-quality terrain rendering. *Journal of Winter School of Computer Graphics*, 14(1-3):49–56.

[Schneider et al., 2005] Schneider, M., Guthe, M., and Klein, R. (2005). Real-time rendering of complex vector data on 3D terrain models. In *Proceedings International Conference on Virtual Systems and Multimedia*, pages 573–582.

[Selassie et al., 2011] Selassie, D., Heller, B., and Heer, J. (2011). Divided edge bundling for directional network data. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2354 – 2363.

[Shanmugam and Arikan, 2007] Shanmugam, P. and Arikan, O. (2007). Hardware accelerated ambient occlusion techniques on gpus. In *Proceedings Symposium on Interactive 3D Graphics and Games*, pages 73–80.

[Shi and Cheung, 2006] Shi, W. and Cheung, C. (2006). Performance evaluation of line simplification algorithms for vector generalization. *The Cartographic Journal*, 43:27–44.

[Shishkovtsov, 2005] Shishkovtsov, O. (2005). Deferred shading in s.t.a.l.k.e.r. In *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional.

[Sun et al., 2008] Sun, M., Lv, G. L., and Lei, C. (2008). Large-scale vector data displaying for interactive manipulation in 3D landscape map. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 37:507–512.

[Thatcher, 2002] Thatcher, U. (2002). Super-size it! scaling up to massive virtual worlds. In *ACM SIGGRAPH 2002 Courses*.

[Thöny et al., 2015] Thöny, M., Billeter, M., and Pajarola, R. (2015). The future of scientific terrain visualization. In *Proceedings ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 13:1–13:4.

[Thöny et al., 2016] Thöny, M., Billeter, M., and Pajarola, R. (2016). Deferred vector map visualization. In *Proceedings ACM SIGGRAPH Asia Symposium on Visualization*, pages 16:1–16:8.

[Thöny and Pajarola, 2013] Thöny, M. and Pajarola, R. (2013). GPU based graph bundling using geographic reference information. In *ACM SIGGRAPH Posters*, pages 111:1–111:1.

[Thöny and Pajarola, 2014] Thöny, M. and Pajarola, R. (2014). GPU accelerated chart visualization in GIS using point splatting. In *Extended Abstract Proceedings GIScience*, pages 424–426.

[Thöny and Pajarola, 2015] Thöny, M. and Pajarola, R. (2015). Vector map constrained path bundling in 3D environments. In *Proceedings ACM SIGSPATIAL International Workshop on GeoStreaming*, pages 33–42.

[Treib et al., 2012] Treib, M., Reichl, F., Auer, S., and Westermann, R. (2012). Interactive editing of gigasample terrain fields. *Computer Graphics Forum*, 31:383–392.

[Vaaraniemi et al., 2011] Vaaraniemi, M., Treib, M., and Westermann, R. (2011). High-quality cartographic roads on high-resolution DEMS. *Journal of Winter School of Computer Graphics*, 19:41–48.

[Wang et al., 2009] Wang, X., Liu, J., and Bi, J. (2009). Rendering of vector data on 3D virtual landscapes. In *Proceedings IEEE International Conference on Information Science and Engineering*, pages 2125–2128.

[Wartell et al., 2003] Wartell, Z., Kang, E., Wasilewski, T., Ribarsky, W., and Faust, N. (2003). Rendering vector data over global, multi-resolution 3D terrain. In *Proceedings Eurographics Symposium on Data Visualization*, pages 213–222.

[Wilkie et al., 2012] Wilkie, D., Sewall, J., and Lin, M. C. (2012). Transforming gis data into functional road models for large-scale traffic simulation. *IEEE Transactions on Visualization and Computer Graphics*, 18(6):890–901.

[Wimmer and Bittner, 2005] Wimmer, M. and Bittner, J. (2005). Hardware occlusion queries made useful. In *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional.

[Wong and Carpendale, 2005] Wong, N. and Carpendale, S. (2005). Interactive poster: Using edge plucking for interactive graph exploration. In *Proceedings IEEE Symposium in Information Visualization - Posters*.

[Xu et al., 2010] Xu, Y., Sui, Z., Weng, J., and Ji, X. (2010). Visualization methods of vector data on a Digital Earth System. In *Proceedings International Conference on Geoinformatics*, pages 1–5.

[Yang et al., 2011] Yang, L., Zhang, L., Ma, J., Kang, Z., Zhang, L., and Li, J. (2011). Efficient simplification of large vector maps rendered onto 3D landscapes. *IEEE Computer Graphics and Applications*, 31(2):14–23.

[Zielasko et al., 2016] Zielasko, D., Weyers, B., Hentschel, B., and Kuhlen, T. (2016). Interactive 3d force-directed edge bundling. *Computer Graphics Forum*, 35(3):51–60.

[Zinsmaier et al., 2012] Zinsmaier, M., Brandes, U., Deussen, O., and Strobelt, H. (2012). Interactive level-of-detail rendering of large graphs. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2486 –2495.

CURRICULUM VITAE

Personal Information

Name	Matthias Thöny
Date of birth	February 17, 1984
Place of birth	Feldkirch, Austria

Education

2011 - 2017	Doctor of Informatics, Visualization and Multimedia Lab, Departement of Informatics, University of Zurich, Switzerland ”Interactive Visualization of Large-Scale Geographic Data” Advisor: Prof. Dr. Renato Pajarola, University of Zurich Examiner: Prof. Dr. Michael Böhlen, University of Zurich Examiner: Prof. Dr. Lorenz Hurni, ETH Zurich
2007 - 2010	MSc. Computer graphics and digital Image Processing, Faculty of Informatics, Technical University of Vienna, Austria
2003 - 2007	BSc. Multimedia Computer Science, Faculty of Informatics, Technical University of Vienna, Austria
1995 - 2003	Liechtensteinisches Gymnasium Vaduz, Liechtenstein

Professional Experience

2011 - 2017	Research assistant, VMML Departement of Informatics, University of Zurich, Switzerland Binzmühlestrasse 14, 8050 Zurich, Switzerland
2010 - 2011	IT-Consultant, Fabasoft Schweiz AG Spitalgasse 36, 3011 Bern, Switzerland
2009 - 2010	Diplomand, EADS Deutschland GmbH Claude-Dornier-Strasse, 88090 Immenstaad, Germany

Publications

Conference Publications

M. Thöny, M. Billeter and R. Pajarola. Deferred vector map visualization. *In Proceedings SIGGRAPH Asia Symposium on Visualization*, pp. 16:1 –16:8, 2016.

M. Thöny, M. Billeter and R. Pajarola. The future of scientific terrain visualization. *In Proceedings ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 13:1 –13:4, 2015.

M. Thöny and R. Pajarola. Vector map constrained path bundling in 3D environments. *In Proceedings ACM SIGSPATIAL International Workshop on GeoStreaming*, pp. 33 –42, 2015.

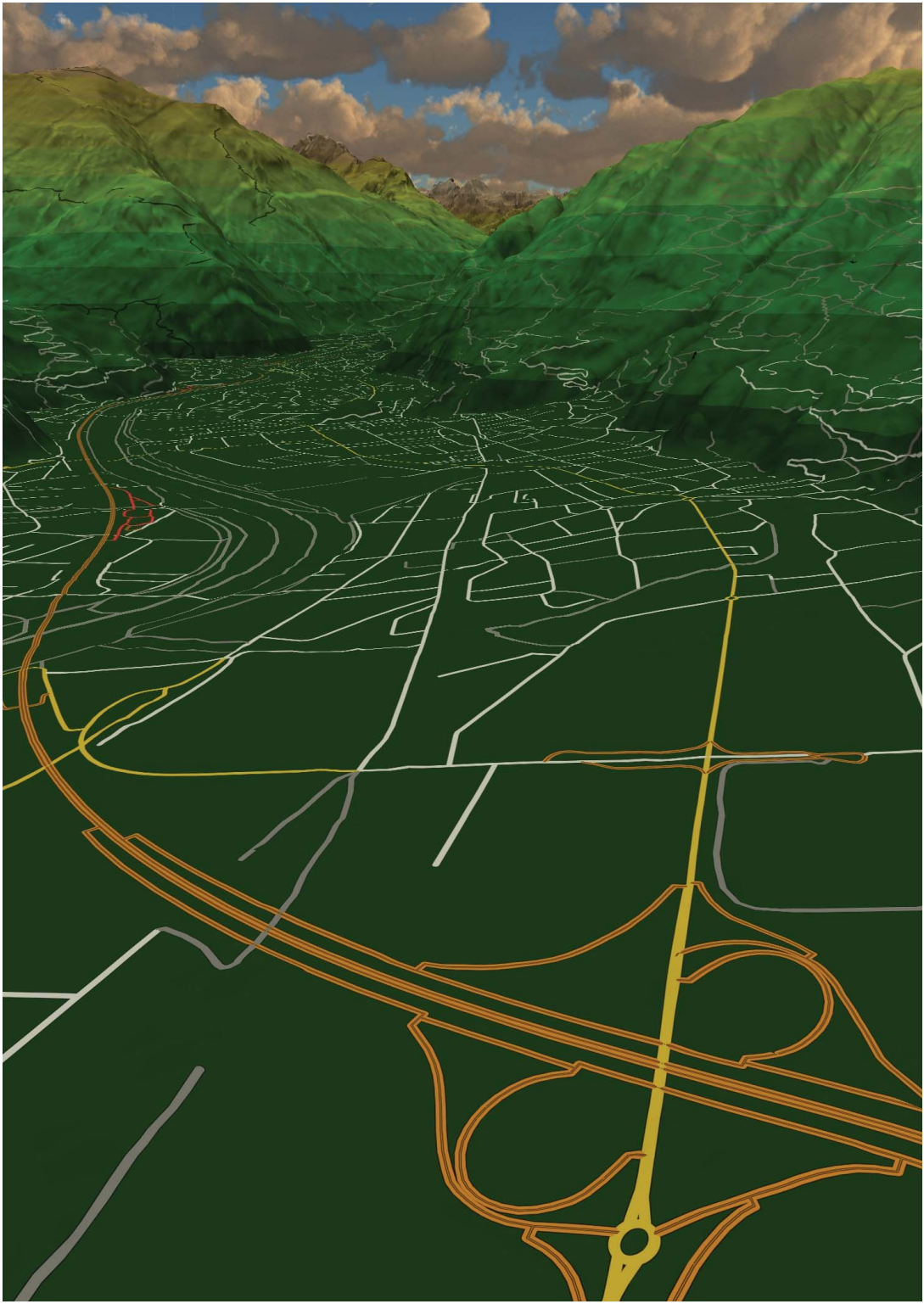
Misc

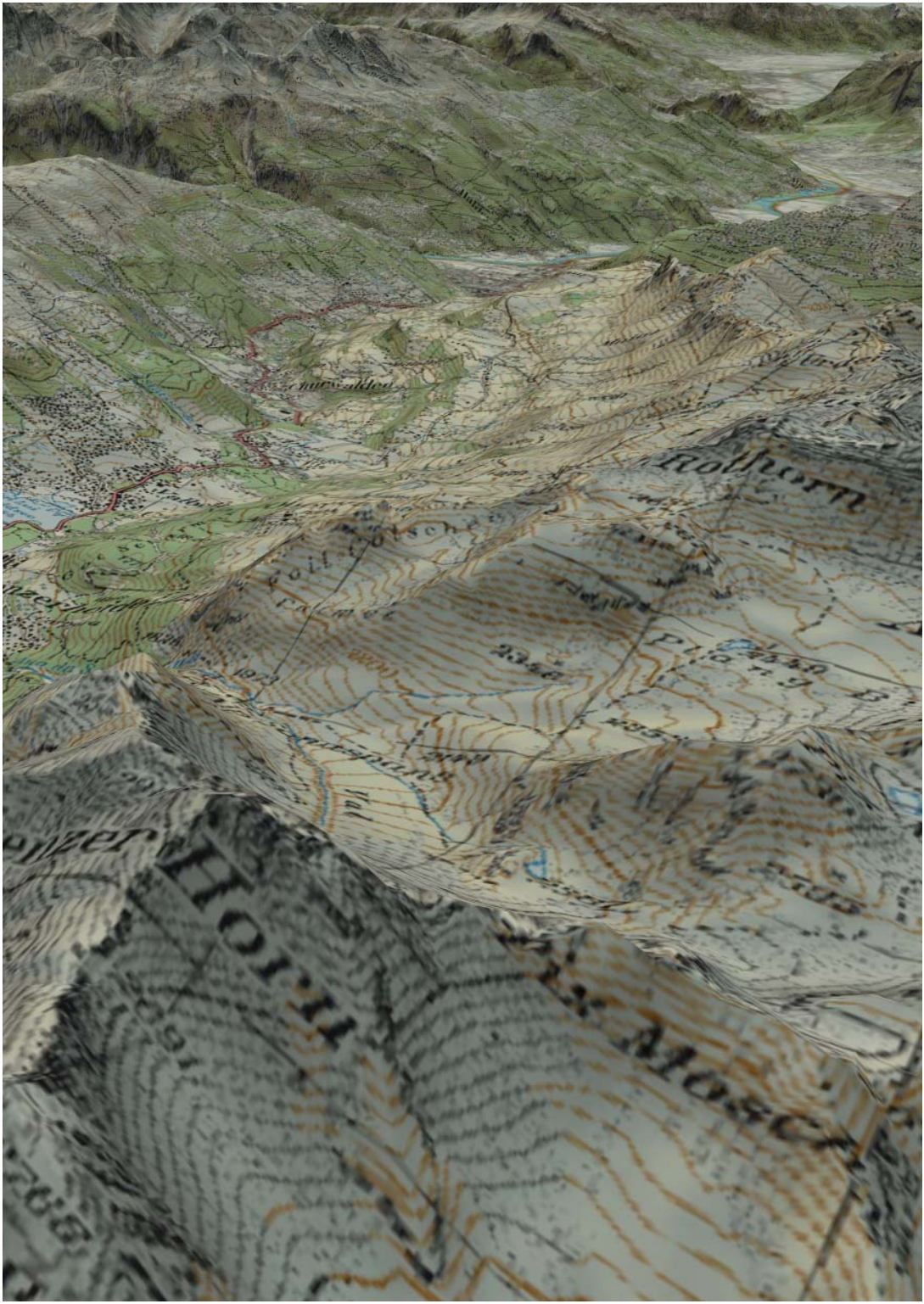
M. Thöny. Interactive Visualization of Large Scale Feature Data in Geographic Information Systems. *IEEE VIS Doctoral Colloquium*, 2014.

M. Thöny and R. Pajarola. GPU accelerated chart visualization in GIS using point splatting. *In Extended Abstract Proceedings of the GIScience*, pp. 424 –426, 2014.

M. Thöny and R. Pajarola. GPU based graph bundling using geographic reference information. *In ACM SIGGRAPH Posters*, pp. 111:1 –111:1, 2013.

M. Thöny. Certifyability of methods to visualise sensor data for synthetic vision systems. Diploma Thesis, TU Wien, 2010.




Rothorn
Piz Nair
Muottas Muragl